Handbuch

TC3 C++

TwinCAT 3

Version: 1.0
Datum: 02.10.2015
Bestell-Nr.: TC1300

BECKHOFF

Inhaltsverzeichnis

1 Vorwort 6
1.1 Hinweise zur Dokumentation 6
1.2 Sicherheitshinweise 7

2 Übersicht 8

3 Einleitung 9

4 Anforderungen 12

5 Vorbereitung - nur einmal! 13
5.1 „Microsoft Windows Driver Kit (WDK)“ Installation 13
5.2 Visual Studio - TwinCAT XAE Base-Symbolleiste 15
5.3 Visual Studio vorbereiten - Konfigurations- und Plattformsymbolleiste 16
5.4 x64: Treibersignierung 16
5.4.1 Treiber signieren 17
5.4.2 Testsignierung freigeben 17
5.4.3 Testzertifikat löschen 19
5.4.4 Kundenzertifikate 21

6 Module 23
6.1 TwinCAT-Modul 23
6.2 TwinCAT Component Object Model (TcCOM) 26
6.3 Zustandsmaschine 27
6.4 Klassenbeschreibungsdateien 29

7 Module - Handhabung 31
7.1 Module exportieren 31
7.2 Module importieren 32

8 Entwicklung 36

9 Schnellstart 37
9.1 TwinCAT 3 Projekt erstellen 37
9.2 TwinCAT 3 C++ Projekt erstellen 39
9.3 TwinCAT 3 C++ Projekt implementieren 43
9.4 Compile/Build a TwinCAT 3 C++ project 44
9.5 TwinCAT 3 C++ Modulinstanz erstellen 46
9.6 Einen Task erstellen, um eine TwinCAT 3 C++ Modulinstanz aufzurufen 48
9.7 TwinCAT 3 C++, C++ Debugger aktivieren 51
9.8 TwinCAT 3 C++ Projekt aktivieren 52
9.9 TwinCAT 3 C++ Projekt debuggen 53

10 Debuggen 60
10.1 Einzelheiten zu den bedingten Haltepunkten 62

11 Assistenten 65
11.1 TwinCAT C++ Projekt-Assistent 65
11.2 TwinCAT Module Klassenassistent 66
11.3 TwinCAT Module Class Editor (TMC) 69
11.3.1 Übersicht 71
11.3.2 Grundlegende Informationen 71
11.3.3 Datentypen 72
11.3.4 Module 90

11.4 TwinCAT Module Instance Configurator ... 112
11.4.1 Objekt ... 113
11.4.2 Kontext ... 114
11.4.3 Parameter (Init) ... 114
11.4.4 Data Area ... 115
11.4.5 Schnittstellen ... 115
11.4.6 Schnittstellenzeiger ... 115
11.4.7 Datenzeiger ... 116

12 Programmierreferenz ... 117
12.1 Dateibeschreibung ... 117
12.2 Limitierungen ... 119
12.3 Speicherallozierung ... 120
12.4 Schnittstellen ... 120
12.4.1 Schnittstelle ITcUnknown ... 121
12.4.2 Schnittstelle ITComObject ... 123
12.4.3 Schnittstelle ITcCyclic ... 130
12.4.4 Schnittstelle ITcCyclicCaller ... 131
12.4.5 Schnittstelle ITcFileAccess ... 133
12.4.6 Schnittstelle ITcFileAccessAsync ... 141
12.4.7 Schnittstelle ITcPostCyclic ... 143
12.4.8 Schnittstelle ITcPostCyclicCaller ... 144
12.4.9 Schnittstelle ITcIoCyclic ... 146
12.4.10 Schnittstelle ITcIoCyclicCaller ... 147
12.4.11 Schnittstelle ITcRTimeTask ... 149
12.4.12 Schnittstelle ITcTask ... 150
12.4.13 Schnittstelle ITcTaskNotification ... 154
12.5 ADS-Kommunikation ... 155
12.5.1 AdsReadDeviceInfo ... 156
12.5.2 AdsRead ... 158
12.5.3 AdsWrite ... 160
12.5.4 AdsReadWrite ... 162
12.5.5 AdsReadState ... 164
12.5.6 AdsWriteControl ... 166
12.5.7 AdsAddDeviceNotification ... 168
12.5.8 AdsDelDeviceNotification ... 170
12.5.9 AdsDeviceNotification ... 172
12.6 Mathematische Funktionen ... 173
12.7 Zeitfunktionen ... 175
12.8 STL / Container ... 175
12.9 Fehlermeldungen - Verständnis ... 176
12.10 Meldungen aufzeichnen ... 176

13 How to...? ... 180
13.1 Ziellaufzeit auswählen ... 180
13.2 Verwenden Sie das Automation Interface ... 184
13.3 Bereitstellung ... 184
13.4 Module veröffentlichen ... 185
13.5 Clone ... 186
13.6 Zugriff auf Variablen über ADS ... 187
13.7 TcCallAfterOutputUpdate für C++ Module ... 187
13.8 Reihenfolgebestimmung der Ausführung in einem Task ... 187
13.9 Stack Size > 4kB verwenden ... 188
13.10 Siehe Version/Herstellerinformation ... 188
13.11 Modul löschen ... 189

13.12 Initialisierung von TMC-Membervariablen ... 190
13.13 SPS-Zeichenketten als Methodenparameter verwenden ... 190
13.14 Bibliotheken von Drittanbietern ... 191

14 Fehlersuche ... 192
14.1 Build - „Cannot open include file ntddk.h“ ... 192
14.2 Build - „The target ... does not exist in the project“ ... 192
14.3 Debug - „Unable to attach“ ... 193
14.4 Activation – „invalid object id“ (1821/0x71d) ... 194
14.5 Fehlermeldung – VS2010 und LNK1123/COFF ... 194
14.6 Verwendung von C++ Klassen in TwinCAT C++ Modulen ... 194

15 C/C++ Beispiele ... 196
15.1 Übersicht ... 196
15.2 Beispiel01: Zyklisches Modul mit IO ... 198
15.3 Beispiel02: Zyklische C++ Logik, die IO vom IO Task verwendet ... 199
15.4 Beispiel03: C++ als ADS Server ... 200
15.4.1 Beispiel03: Der in C++ geschriebene TC3 ADS Server ... 201
15.4.2 Beispiel03: ADS Client UI in C# ... 205
15.5 Beispiel05: C++ CoE Zugriff über ADS ... 209
15.6 Beispiel06: UI-C#-ADS Client lädt die Symbolik vom Modul hoch ... 210
15.7 Beispiel07: Empfang von ADS Notifications ... 215
15.8 Beispiel10: Modulkommunikation: Verwendung von Datenzeigern ... 216
15.9 Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf ... 217
15.9.1 Beispiel11: Eine Methode zur Verfügung stellendes TwinCAT 3 C++ Modul ... 218
15.9.2 Beispiel11: SPS um eine Methode aufzurufen, die von einem anderen Modul angeboten wird (hier TwinCAT 3 C++ Modul) ... 231
15.10 Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an ... 247
15.11 Beispiel12: Modulkommunikation: Verwendet IO Mapping ... 248
15.12 Beispiel19: Synchroner Dateizugriff ... 248
15.13 Beispiel20: FileIO-Write ... 249
15.14 Beispiel20a: FileIO-Cyclic Read / Write ... 250
15.15 Beispiel22: Automation Device Driver (ADD): Zugang DPRAM ... 251
15.16 Beispiel25: Statische Bibliothek ... 252
15.17 Beispiel26: Ausführungsreihenfolge in einem Task ... 254
15.18 Beispiel30: Zeitmessung ... 256
15.19 Beispiel31: Funktionsbaustein TON in TwinCAT3 C++ ... 257
15.20 Beispiel35: Ethernet Zugriff ... 257
15.21 Beispiel37: Daten archivieren ... 259

16 Anhang ... 260
16.1 ADS Return Codes ... 260

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Deshalb ist die Dokumentation nicht in jedem Fall vollständig auf die Übereinstimmung mit den beschriebenen Leistungsdaten, Normen oder sonstigen Merkmalen geprüft.
Falls sie technische oder redaktionelle Fehler enthält, behalten wir uns das Recht vor, Änderungen jederzeit und ohne Ankündigung vorzunehmen.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC®und XTS® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, DE102004044764, DE102007017835
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

Die TwinCAT Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP0851348, US6167425 mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizensiert durch die Beckhoff Automation GmbH, Deutschland

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

GEFAHR	**Akute Verletzungsgefahr!** Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!
WARNUNG	**Verletzungsgefahr!** Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!
VORSICHT	**Schädigung von Personen!** Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!
Achtung	**Schädigung von Umwelt oder Geräten** Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.
Hinweis	**Tipp oder Fingerzeig** Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

2 Übersicht

Dieses Kapitel behandelt die Implementierung von TwinCAT 3 in C/C++. Die wichtigsten Kapitel sind:

- Grundlagen
  Welche Plattformen werden unterstützt? Sind weitere Programme nötig, um TwinCAT 3 C++ Module zu implementieren?
  Die Antworten finden Sie in Anforderungen [▶ 12] und Vorbereitung [▶ 13]. Limitierungen sind hier [▶ 119] dokumentiert.
- Schnellstart [▶ 37]
  Dies ist ein „weniger als fünf Minuten in Anspruch nehmendes Beispiel“ um einen einfachen, zyklisch ausgeführten Zähler in C++ zu implementieren. Der Zählerwert wird überwacht und überschrieben, es sind Debugging-Möglichkeiten beschrieben, usw.
- MODULE [▶ 23] **- MODULE – MODULE**
  Modularisierung ist die Grundphilosophie von TwinCAT 3. Insbesondere für C++ Module ist ein Verständnis des modularen Konzepts von TwinCAT 3 die Grundvoraussetzung.
  Grundkenntnisse zur Architektur von TwinCAT-Modulen sind erforderlich. [▶ 23]
- Assistenten [▶ 65]
  Dokumentation der visuellen Komponenten der TwinCAT C++ Umgebung.
  Hierzu gehören Tools zum Erstellen von Projekten und Tools für die Bearbeitung von Modulen und die Konfiguration von Modulinstanzen.
- Programmierreferenz [▶ 117]
  Dieses Kapitel enthält ausführliche Informationen zur Programmierung in TwinCAT C++. Hier findet man Schnittstellen und andere TwinCAT-Funktionen für ADS-Kommunikation und Hilfsmethoden.
- Das „How to ...? [▶ 180]“ Kapitel enthält nützliche Tipps für das Arbeiten mit TwinCAT C++.
- Beispiele [▶ 196]
  Einige Schnittstellen und deren Verwendung sind ausführlich in Form eines ausführbaren Programms beschrieben, das als Download mit Quellcode und Solution zur Verfügung gestellt wird.

Sehen Sie dazu auch

- Module exportieren [▶ 31]
- Einleitung [▶ 9]
- Schnittstellen [▶ 120]

3 Einleitung

Die Methode, klassische Automatisierungsgeräte wie speicherprogrammierbare Steuerungen (SPS) und numerische Steuerungen (NC) als Software auf leistungsstarker Standard-Hardware nachzubilden, ist seit vielen Jahren der Stand der Technik und wird nun von vielen Herstellern praktiziert.

Es gibt viele Vorteile, aber der wichtigste ist sicher, dass die Software weitgehend hardwareunabhängig ist. Das heißt zum einen, dass die Leistungsfähigkeit der Hardware speziell an die Anwendung angepasst werden kann und zum anderen, dass man automatisch von ihrer Weiterentwicklung profitieren kann.

Dies gilt insbesondere für PC-Hardware, deren Leistung noch immer dramatisch schnell ansteigt. Die relative Unabhängigkeit von einem Lieferanten, die aus dieser Trennung von Software und Hardware resultiert, ist für den Nutzer auch sehr wichtig.

Da SPS und Bewegungssteuerung - und möglicherweise andere Automatisierungskomponenten - bei dieser Methode unabhängige Logikbausteine bleiben, gibt es im Vergleich zur klassischen Automatisierungstechnologie nur wenige Änderungen in der Anwendungsarchitektur.

Die SPS bestimmt die logischen Abläufe der Maschine und überträgt der Bewegungssteuerung die Implementierung bestimmter Achsfunktionen. Dank der verbesserten Leistung der Steuerungen und der Möglichkeit, höhere Programmiersprachen (IEC 61131-3) zu verwenden, können auf diese Weise auch komplexe Maschinen automatisiert werden.

Modularisierung

Um die Komplexität moderner Maschinen zu meistern und gleichzeitig den dafür nötigen Engineering-Aufwand zu senken, haben viele Hersteller begonnen, ihre Maschinen zu modularisieren. Individuelle Funktionen, Baugruppen oder Maschineneinheiten werden dabei als Module betrachtet, die untereinander so unabhängig wie möglich sind und über einheitliche Schnittstellen in das Gesamtsystem eingebettet werden.

Eine Maschine ist danach idealerweise hierarchisch strukturiert, wobei Module der niedrigsten Ebene einfachste, ständig wiederverwendbare Grundelemente repräsentieren. Miteinander verbunden bilden sie immer komplexere Maschineneinheiten, bis auf höchster Ebene die gesamte Maschine entsteht. Für Steuerungssysteme bei Maschinenmodularisierung gibt es unterschiedliche Herangehensweisen. Sie können grob in eine dezentralisierte und eine zentralisierte Vorgehensweise unterteilt werden.

Beim lokalen Ansatz hat jede Maschine ihre eigene Steuerung, die die SPS- und möglicherweise auch die Bewegungsfunktionen des Moduls übernimmt.

Die individuellen Module können getrennt voneinander in Betrieb genommen und gewartet werden. Sie können relativ unabhängig voneinander skaliert werden. Die notwendigen Interaktionen zwischen den Steuerungen werden über Kommunikationsnetzwerke (Feldbusse oder Ethernet) koordiniert und über geeignete Profile standardisiert.

Der zentralisierte Ansatz konzentriert die Steuerungsfunktionen aller Module in einer gemeinsamen Steuerung und nutzt die Rechenintelligenz der lokalen I/O-Geräte nur sehr wenig. Interaktionen können in der zentralen Steuerungseinheit viel direkter ausgeführt werden, da die Kommunikationswege viel kürzer sind. Es treten keine Totzeiten auf und die Steuerungshardware wird viel besser ausgenutzt, was die Gesamtkosten senkt.

Allerdings hat die zentralisierte Methode auch den Nachteil, dass eine Modularisierung der Steuerungssoftware nicht automatisch notwendig ist. Die Möglichkeit, auf alle Informationen anderer Programmteile in der zentralen Steuerung zugreifen zu können, bietet keinen Anreiz zum Aufbau der Steuerungssoftware aus Modulen, die in anderen Anwendungen wiederverwendet werden können. Da es zwischen den Steuerungseinheiten keinen Kommunikationskanal gibt, bleiben die Entwicklung eines geeigneten Profils und die Standardisierung der Steuerungseinheiten häufig auf der Strecke.

Das Beste aus beiden Welten

Die ideale Steuerung für modulare Maschinen nutzt Elemente aus dezentralisierter und zentralisierter Steuerungsarchitektur. Eine zentrale, leistungsstarke und möglichst allgemeine Computerplattform dient „wie üblich“ als Steuerungshardware.

Die Vorteile einer zentralisierten Steuerungstechnologie:

- geringe Gesamtkosten
- verfügbar
- schnelles, modulares Feldbussystem (Stichwort EtherCAT)
- und die Möglichkeit, auf alle Informationen im System ohne Kommunikationsverlust zugreifen zu können

sind entscheidende Argumente.

Die oben aufgeführten Vorteile einer dezentralisierten Herangehensweise können in der zentralisierten Steuerung durch geeignete Modularisierung der Steuerungssoftware umgesetzt werden.

Anstelle ein großes, komplexes SPS-Programm und eine NC mit vielen Achsen ablaufen zu lassen, können viele kleine „Steuerungen“ in einer gemeinsamen Laufzeit nebeneinander auf der gleichen Hardware relativ unabhängig voneinander koexistieren. Die einzelnen Steuerungsmodule sind in sich abgeschlossen und stellen der Umgebung ihre Funktionen über Standard-Schnittstellen zur Verfügung oder nutzen entsprechende Funktionen anderer Module oder der Laufzeit.

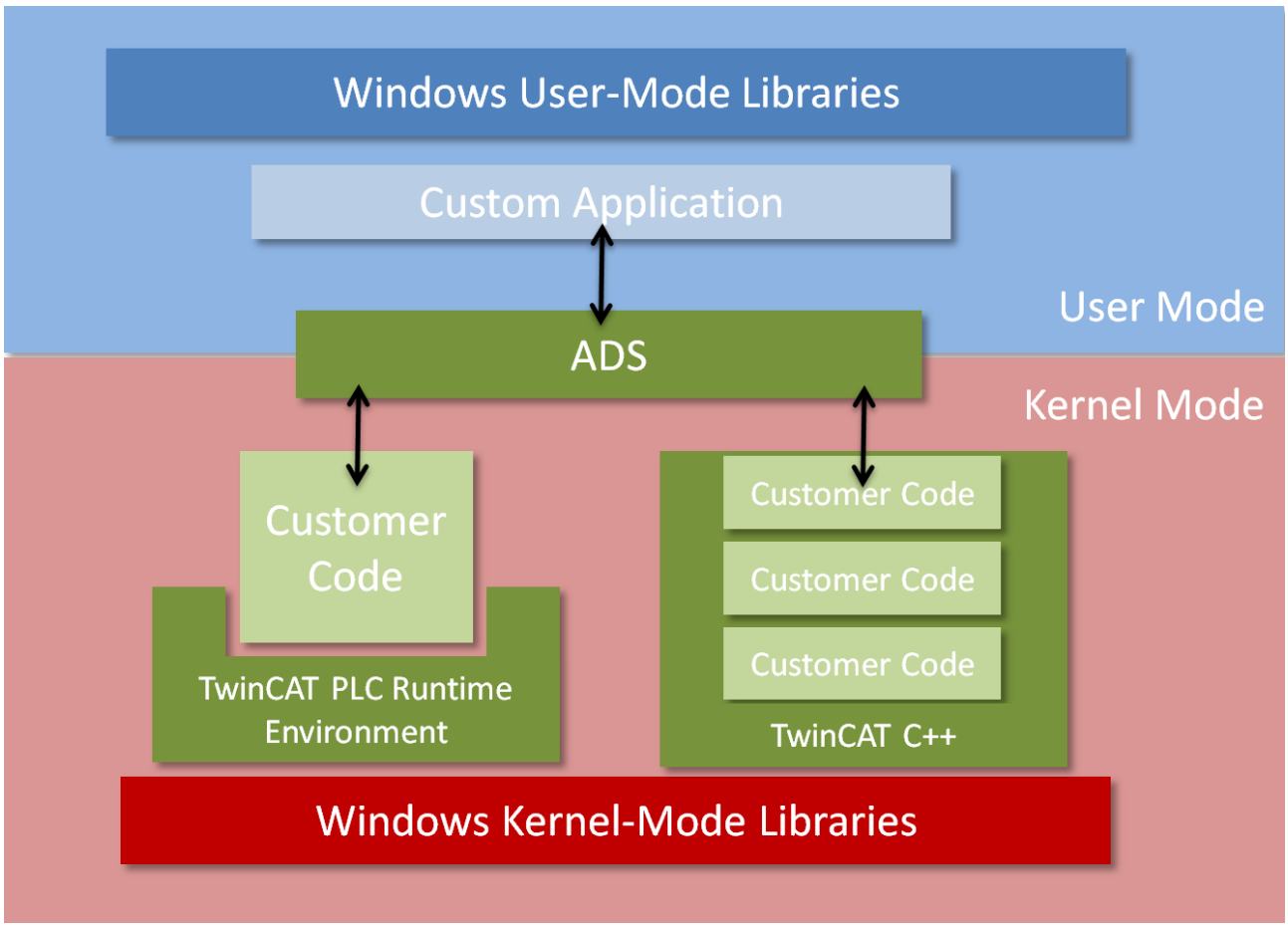
Durch die Definition dieser Schnittstellen und die Standardisierung der entsprechenden Parameter und Prozessdaten entsteht ein signifikantes Profil. Da die einzelnen Module in eine Laufzeit implementiert sind, sind auch direkte Aufrufe anderer Module möglich - wiederum über entsprechende Standard-Schnittstellen. Die Modularisierung kann so in sinnvollen Grenzen stattfinden, ohne dass Kommunikationsverluste ein Thema sind.

Während der Entwicklung oder Inbetriebnahme einzelner Maschinenmodule können die entsprechenden Steuerungsmodule erstellt und auf beliebiger Steuerungshardware mit geeigneter Laufzeit getestet werden. Fehlende Anbindungen an andere Module können in dieser Phase emuliert werden. Bei der kompletten Maschine werden sie dann zusammen auf der zentralen Laufzeit instanziiert, die lediglich so dimensioniert sein muss, dass der Bedarf an Ressourcen für alle instanziierten Module (Speicher, Tasks und Rechenleistung) erfüllt wird.

TwinCAT 3 Runtime

TwinCAT Runtime bietet eine Softwareumgebung in der die TwinCAT Module geladen, implementiert und verwaltet werden. Sie bietet weitere Basisfunktionen, sodass die Systemressourcen (Speicher, Tasks, Feldbus und Hardwarezugang, usw.) genutzt werden können. Die einzelnen Module müssen nicht mit demselben Compiler erstellt worden sein. Damit können sie voneinander unabhängig sein und von unterschiedlichen Anbietern stammen.

Beim Start der Laufzeit werden automatisch einige Systemmodule geladen, sodass ihre Eigenschaften anderen Modulen zur Verfügung stehen. Allerdings erfolgt der Zugriff auf die Eigenschaften der Systemmodule auf die gleiche Weise wie auf die Eigenschaften normaler Module, sodass es für die Module keine Rolle spielt, ob die jeweilige Eigenschaft von einem Systemmodul oder einem normalen Modul zur Verfügung gestellt wird.



Im Gegensatz zur SPS, wo das benutzerdefinierte Programm in einer Laufzeitumgebung ausgeführt wird, sind TwinCAT C++ Module nicht in so einer gehosteten Umgebung. Dadurch werden TwinCAT C++ Module als Kernel-Module (.sys) ausgeführt, sie werden also demzufolge aus Kernel-Modus-Bibliotheken erstellt.

4 Anforderungen

Übersicht der Mindestanforderungen

Die Implementierung und das Debuggen von TwinCAT 3 C++ Modulen erfordert:

Auf dem Engineering PC (PC, der für Erstellung und Bearbeitung des Codes verwendet wird):

- PC mit Microsoft Betriebssystem (Windows XP oder Windows 7), Plattform für x86 (32bit) oder x64 Betriebssystem wird unterstützt
- Microsoft Visual Studio 2010 (mit Service Pack 1), 2012 und 2013
  Level Professional, Premium oder Ultimate
  HINWEIS! Unter Windows 7 bitte VS2010 / 2012 / 2013 mit Rechtsklick auf „Run As Admin…" installieren (Selbst dann, wenn Sie der Admin auf der Maschine sind!)
- Microsoft „Windows Driver Kit" (WDK)
  Wenn Sie C++ Module implementieren und debuggen wollen, muss Microsoft „Windows Driver Kit" (WDK) installiert sein: „Windows Driver Kit" (WDK) installieren [▶ 13]
- TwinCAT 3 Installation (XAE Engineering)

Auf dem Laufzeit-PC:

- IPC oder Embedded CX Rechner mit Microsoft Betriebssystem (Windows XP oder Windows 7 oder höher)
- Microsoft Visual Studio muss nicht installiert werden
- Microsoft „Windows Driver Kit" (WDK) muss nicht installiert werden.
  (Für die Integration und Nutzung bestehender, binärer C++ Module in eine TwinCAT 3 SPS-Umgebung ist keine zusätzliche Installation erforderlich.)
- TwinCAT 3 Installation (XAR Laufzeit)

Limitierungen auf dem Laufzeit-PC

- TwinCAT 3.0 unterstützt 32-Bit-Betriebssysteme nur als Zielplattform (Laufzeit-PC)
  Auf x64 PCs kann TC3.0 als Engineering-Plattform verwendet werden: das Programm kann über Netzwerk auf einen 32bit (x86) Remote-PC übertragen und dort ausgeführt werden.
- TC3.1 unterstützt auch x64-Bit-Betriebssysteme als Zielplattform (Laufzeit-PC)
- Die Ziellaufzeit muss auf „Windows NT Kernel" basieren, wie Windows XP, Windows 7 oder die embedded Versionen wie Windows XP Embedded, Windows Embedded Standard 2009, Windows Embedded Standard 7

5 Vorbereitung - nur einmal!

Ein PC für das Engineering von TwinCAT C++ Modulen muss vorbereitet werden. Diese Schritte müssen nur einmal durchgeführt werden:

- Microsoft Windows Driver Kit (WDK) [▶ 13] muss installiert werden und
- TwinCAT Basis [▶ 15] sowie Konfigurations- und Plattform [▶ 16]-Symbolleiste müssen konfiguriert werden
- Auf x64 PCs müssen Module signiert sein, damit sie ausgeführt werden können.
  Siehe Dokumentation zum Setup einer Testsignierung [▶ 16].

5.1 „Microsoft Windows Driver Kit (WDK)“ Installation

Übersicht

Die Implementierung von TwinCAT 3 C++ Modulen erfordert Teile des „Windows Driver Kit“ (WDK).

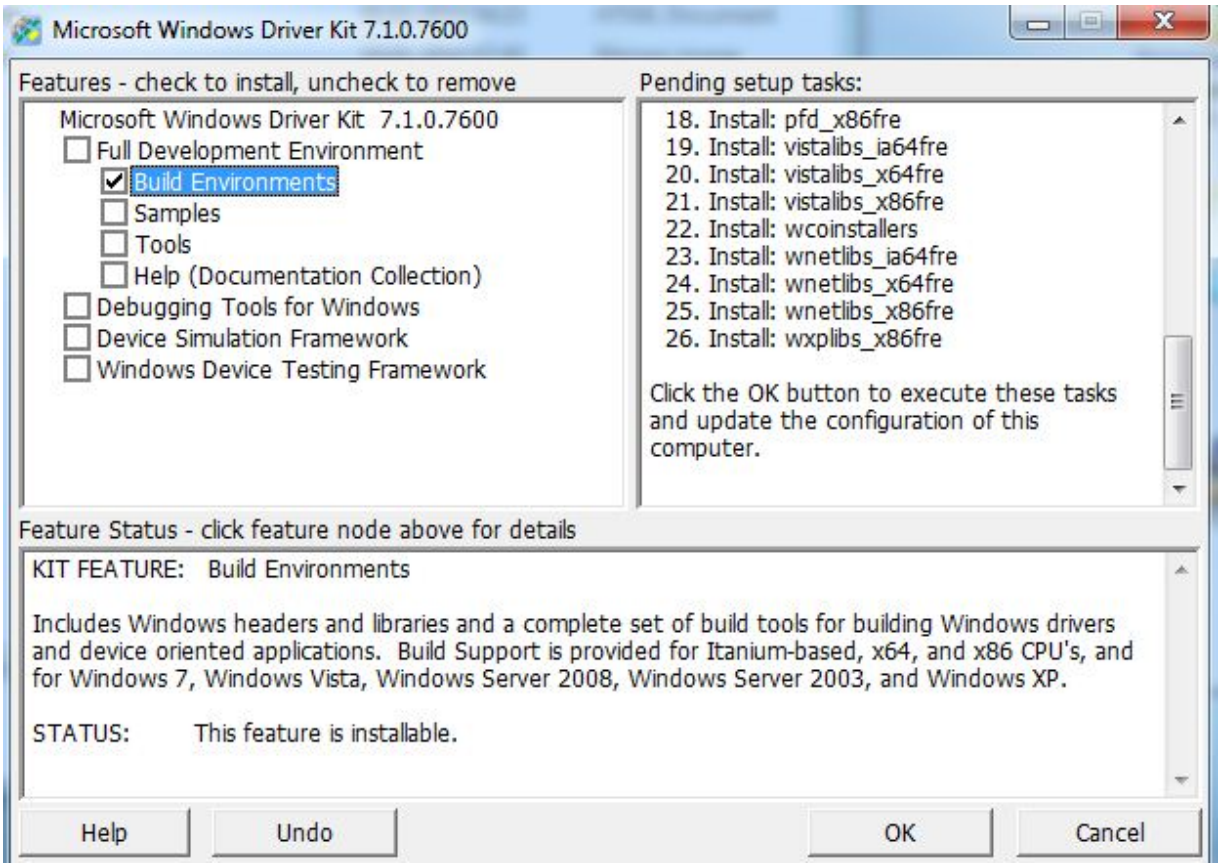
HINWEIS! Die Installation ist nur für die TwinCAT 3 Engineering Umgebung erforderlich, um C++ Module erstellen und bearbeiten zu können. Für die Zielplattform der TwinCAT 3 Runtime wird sie nicht benötigt.

1. „Windows Driver Kit 7.1“ aus dem Microsoft Download Center herunterladen http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=36a2630f-5d56-43b5-b996-7633f2ec14ff

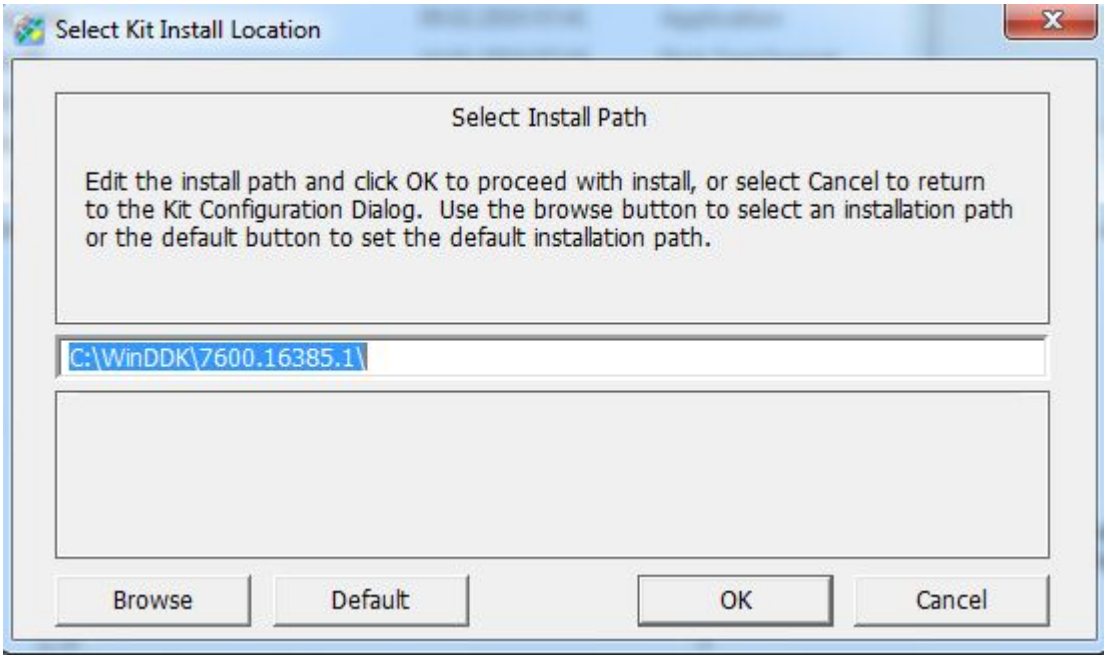
Windows Driver Kit Version 7.1.0
Language:
English
Download
The Windows Driver Kit (WDK) Version 7.1.0 is an update to the WDK 7.0.0 release and contains the tools, code samples, documentation, compilers, headers and libraries with which software developers create drivers for Windows 7, Windows Vista, Windows XP, Windows Server 2008 R2, Windows Server 2008, and Windows Server 2003.
Details
System Requirements
Install Instructions

2. Brennen Sie sich nach dem Download entweder eine CD des heruntergeladenen ISO-Abbilds oder verwenden Sie ein virtuelles (softwarebasiertes) CD-Laufwerk.
3. „KitSetup.exe“ auf CD / heruntergeladenem ISO-Abbild starten (auf Windows7 PCs Installation mit „Run As Administrator...“ starten)

4. Option „Build Environment“ wählen - alle anderen Komponenten werden von TwinCAT 3 nicht benötigt - und „OK“ klicken um fortzufahren.

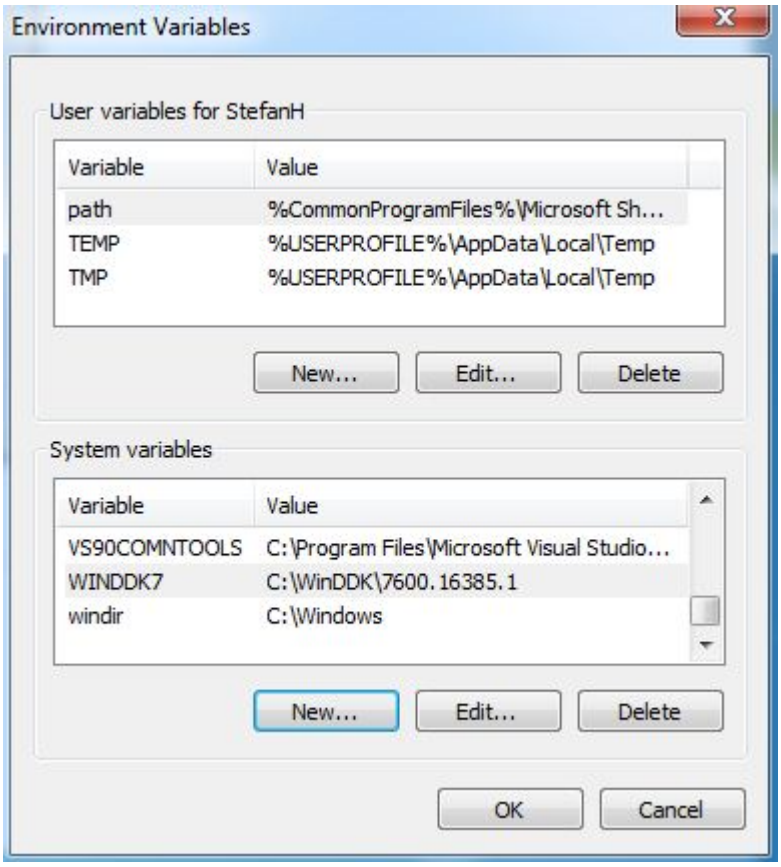

Microsoft Windows Driver Kit 7.1.0.7600
Features - check to install, uncheck to remove
Microsoft Windows Driver Kit 7.1.0.7600
Full Development Environment
Build Environments
Samples
Tools
Help (Documentation Collection)
Debugging Tools for Windows
Device Simulation Framework
Windows Device Testing Framework
Pending setup tasks:
18. Install: pfd_x86fre
19. Install: vistalibs_ia64fre
20. Install: vistalibs_x64fre
21. Install: vistalibs_x86fre
22. Install: wcoinstallers
23. Install: wnetlibs_ia64fre
24. Install: wnetlibs_x64fre
25. Install: wnetlibs_x86fre
26. Install: wxplibs_x86fre
Click the OK button to execute these tasks and update the configuration of this computer.
Feature Status - click feature node above for details
KIT FEATURE: Build Environments
Includes Windows headers and libraries and a complete set of build tools for building Windows drivers and device oriented applications. Build Support is provided for Itanium-based, x64, and x86 CPU's, and for Windows 7, Windows Vista, Windows Server 2008, Windows Server 2003, and Windows XP.
STATUS: This feature is installable.
Help
Undo
OK
Cancel

5. Nach Zustimmung zur Microsoft EULA Lizenz Zielordner für die Installation wählen. Standardmäßig wird der Stammordner **„C:\“** ausgewählt - also wird **„C:\WinDDK\7600.16385.1“** vorgeschlagen
   HINWEIS! Die Ziffern „7600....“ sind im Falle einer neueren Version von „Windows Driver Kit“ möglicherweise andere.
6. Installation mit „OK“ starten


Select Kit Install Location
Select Install Path
Edit the install path and click OK to proceed with install, or select Cancel to return to the Kit Configuration Dialog. Use the browse button to select an installation path or the default button to set the default installation path.
C:\WinDDK\7600.16385.1\
Browse
Default
OK
Cancel

7. Zukünftig übernimmt TwinCAT 3 den folgenden Schritt. Jetzt muss er noch einmal manuell erledigt werden.

8. Navigieren Sie zu „Start“-> „Control Panel“ -> „System“ und wählen „Advanced system settings“
9. Wählen Sie den Karteireiter „Advanced“ und klicken dann auf „Environment Variables...“
10. Im unteren Bereich von „System variables“ wählen Sie „New..“ und fügen folgende Information ein:
    Variablenname WINDDK7
    Variablenwert C:\WinDDK\7600.16385.1
    Der Pfad kann bei einer anderen Version von Windows Driver Kit oder bei Angabe eines anderen Installationspfads abweichen.



11. Nach der Installation neu anmelden oder PC neu starten, um die neuen Umgebungsvariableneinstellungen zu übernehmen.

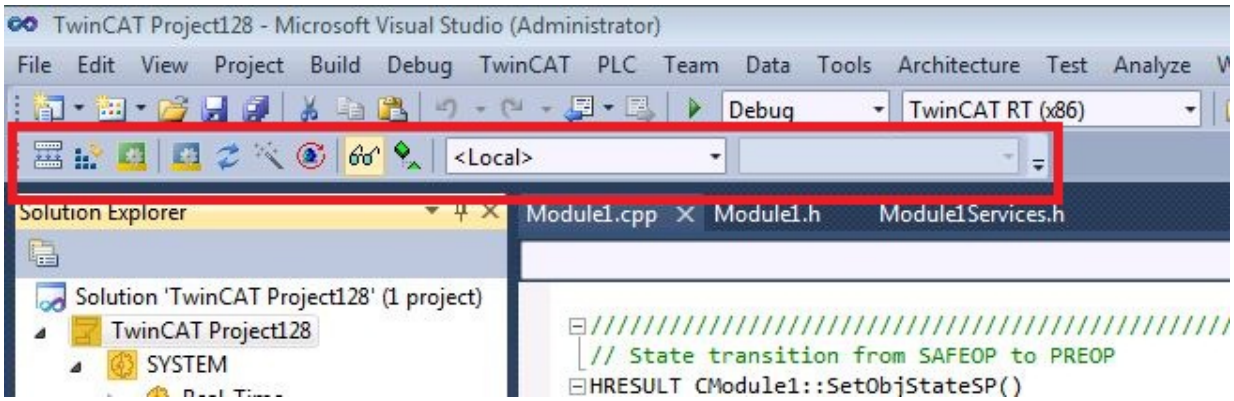
5.2 Visual Studio - TwinCAT XAE Base-Symbolleiste

Fügen Sie für ein effizientes Engineering die „TwinCAT XAE Base“-Symbolleiste hinzu

TwinCAT 3 integriert für eine bessere Effizienz seine eigene Symbolleiste in das Visual Studio-Menü, die Sie bei der Erstellung von C++ Projekten unterstützt. Diese Symbolleiste wird vom TwinCAT 3 Setup automatisch zum Visual Studio-Menü hinzugefügt. Wenn Sie sie allerdings manuell hinzufügen möchten, müssen Sie folgendes tun:

1. Öffnen Sie das Menü „View“ und wählen „Toolbars\TwinCAT XAE Base“

⇨ Die ausgewählte Symbolleiste erscheint unter dem Menü.



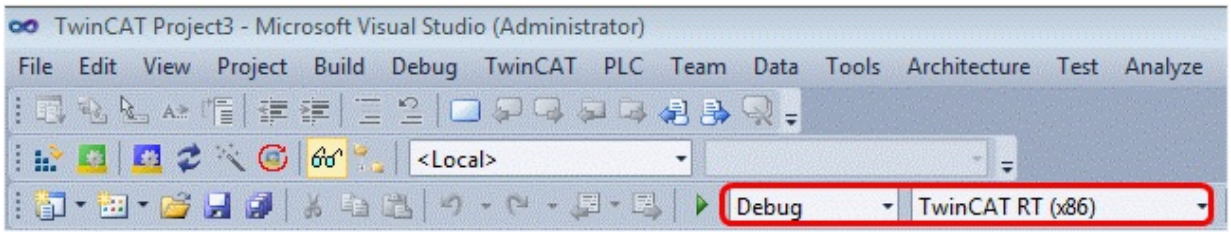
5.3 Visual Studio vorbereiten - Konfigurations- und Plattformsymbolleiste

Symbolleiste „Solution Configuration and Solution Platform“ hinzufügen

Mit der „Configuration and Platform“-Symbolleiste können Sie die Zielplattform für das Erstellen Ihres Projekts bestimmen. Diese Symbolleiste wird vom TwinCAT 3 Setup automatisch zum Visual Studio-Menü hinzugefügt. Wenn Sie sie allerdings manuell hinzufügen möchten, müssen Sie folgendes tun:

1. Öffnen Sie das Menü „View“ und wählen „Toolbars\Customize“
2. Navigieren Sie zum Karteireiter „Commands“
3. Aktivieren Sie das Optionsfeld „Toolbar“ und wählen dann aus der Liste der Symbolleisten „Standard“
4. Klicken Sie auf „Add Command...“
5. Wählen Sie die Kategorie „Build“, wählen den Befehl „Solution Configurations“ aus und klicken dann auf „Ok“.
6. Wiederholen Sie den letzten Schritt für den Befehl „Solution Platforms“
7. Zum Abschluss auf „Close“ klicken

⇨ Nun erscheinen beide Befehle unter der Menüleiste


TwinCAT Project3 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug TwinCAT PLC Team Data Tools Architecture Test Analyze
<Local>
Debug
TwinCAT RT (x86)

5.4 x64: Treibersignierung

Auf x64 PCs müssen TwinCAT C++ Module signiert werden, damit sie ausgeführt werden können.

HINWEIS! Lediglich die Ausführung erfordert Zertifikate - das Engineering auf einer x64-Maschine mit Ausführung auf einer x86-Maschine erfordert keine Signierung.

Da ein veröffentlichtes Modul auf verschiedenen PCs lauffähig sein soll, ist für eine Veröffentlichung („Publish“) immer eine Signierung erforderlich.

5.4.1 Treiber signieren

Übersicht

Für die Implementierung von TwinCAT 3 C++ Modulen auf x64-Plattformen muss der Treiber mit einem Zertifikat signiert sein.

Die Signatur, die beim TwinCAT 3 Build-Prozess automatisch generiert wird, wird von 64-Bit-Windows-Betriebssystemen für die Authentifizierung der Treiber verwendet.

Für die Signierung eines Treibers wird ein Zertifikat benötigt. Diese Microsoft Dokumentation beschreibt den Prozess und Hintergrundwissen zum Erhalt eines Test- und Freigabezertifikats, das von 64-Bit-Windows-Betriebssystemen akzeptiert wird.

Um ein solches Zertifikat in TwinCAT 3 zu verwenden, konfigurieren Sie den Schritt nach Kompilieren Ihres x64 Build-Targets wie in „Ein Testzertifikat für den Testmodus erstellen [▶ 17]“ dokumentiert.

Testzertifikate

Zum Testen können selbstsignierte Testzertifikate ohne technische Limitierung erstellt und verwendet werden.

In den folgenden Lernprogrammen ist beschrieben, wie diese Möglichkeit aktiviert wird.
Um Treiber mit echten Zertifikaten für Produktionsmaschinen zu erstellen, muss diese Möglichkeit deaktiviert werden.

- Ein Testzertifikat für den Testmodus erstellen [▶ 17]
- (Test-)Zertifikate löschen [▶ 19]

Weitere Referenzen:

MSDN, Testzertifikate (Windows Treiber),
http://msdn.microsoft.com/en-us/library/windows/hardware/ff553457(v=vs.85).aspx

MSDN, MakeCert Testzertifikate (Windows Treiber),
http://msdn.microsoft.com/en-us/library/windows/hardware/ff548693(v=vs.85).aspx

5.4.2 Testsignierung freigeben

Übersicht

Für die Implementierung von TwinCAT 3 C++ Modulen auf x64-Plattformen muss der Treiber mit einem Zertifikat signiert sein.

Dieser Artikel beschreibt, wie Sie ein Testzertifikat zum Testen eines C++ Treibers erstellen und installieren können.

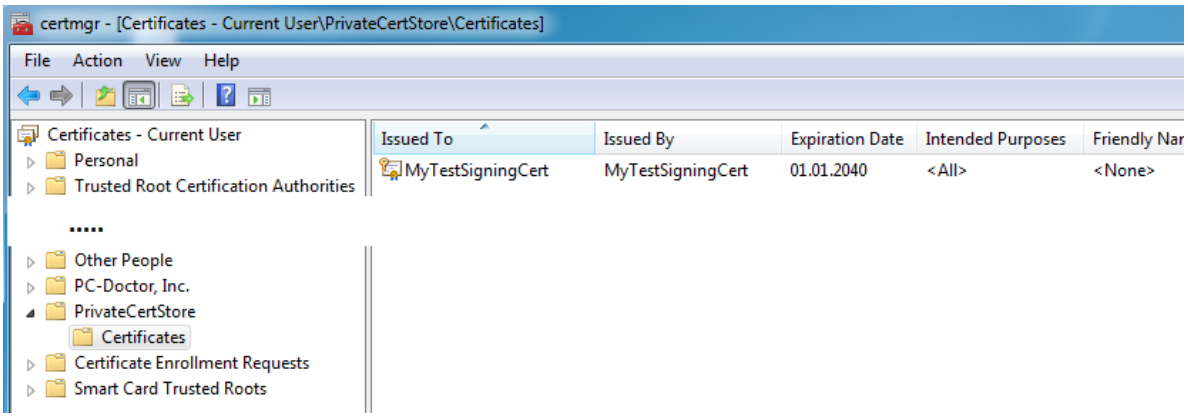
HINWEIS! Entwickler verfügen möglicherweise über eine Vielzahl an Werkzeugen, um selber Zertifikate zu erstellen - bitte folgen Sie dieser Beschreibung genauestens, um den Testzertifikat-Mechanismus anzuschalten.

Einer der folgenden Befehle muss ausgeführt werden

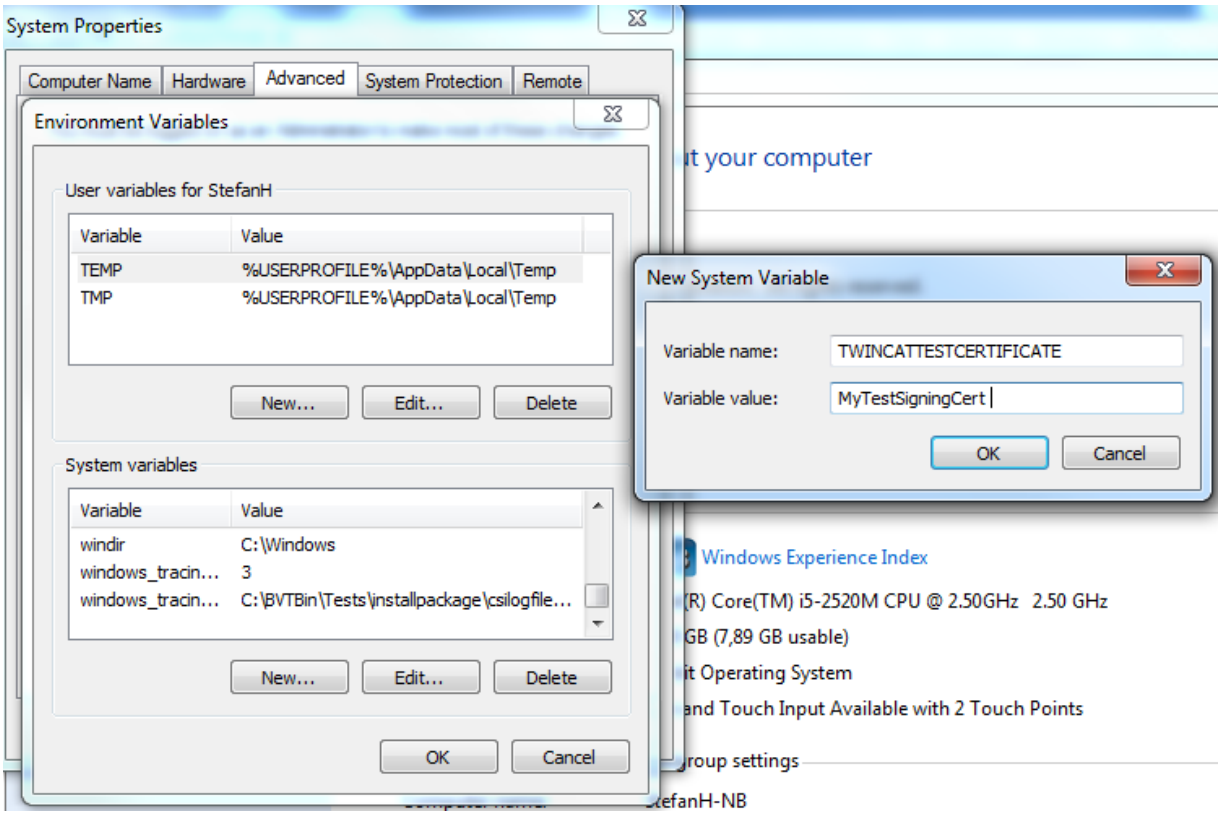
- **Eingabeaufforderung von Visual Studio 2010 / 2012 mit Administratorrechten**. (Über: All Programs -> Microsoft Visual Studio 2010/2012 -> Visual Studio Tools -> Visual Studio Command Prompt, dann Rechtsklick auf „Run as administrator“)

- normale Eingabeaufforderung (Start->Command Prompt) **mit Administratorrechten**, dann ins Verzeichnis %WINDDK7/%\\bin\\x86\ wechseln, das die entsprechenden Tools enthält.

1. Auf XAE:
   Im Engineering System folgenden Befehl in die Visual Studio 2010 / 2012 Eingabeaufforderung mit Administratorrechten eingeben (siehe Hinweis oben):
   ```
   makecert -r -pe -ss PrivateCertStore -n CN=MyTestSigningCert
   MyTestSigningCert.cer
   ```
   ⇨ Daraufhin wird ein selbstsigniertes Zertifikat erstellt und in der Datei „MyTestSigningCert.cer“ und im Windows Certificate Store gespeichert
   ⇨ Das Ergebnis kann mit mmc (Use File->Add/Remove Snap-in->Certificates) überprüft werden:



2. Auf XAE:
   Zertifikat konfigurieren, sodass es von TwinCAT XAE auf dem Engineering System erkannt wird. Umgebungsvariable **„TWINCATTESTCERTIFICATE“** auf **„MyTestSigningCert“** im Engineering System setzen oder jeweils den Post Build Step von **„Debug|TwinCAT RT (x64)“** und **„Release|TwinCAT RT (x64)“** bearbeiten.
   Der Name der Variablen ist NICHT der Name der Zertifikatsdatei, sondern der CN-Name (in diesem Falle MyTestSigningCert).



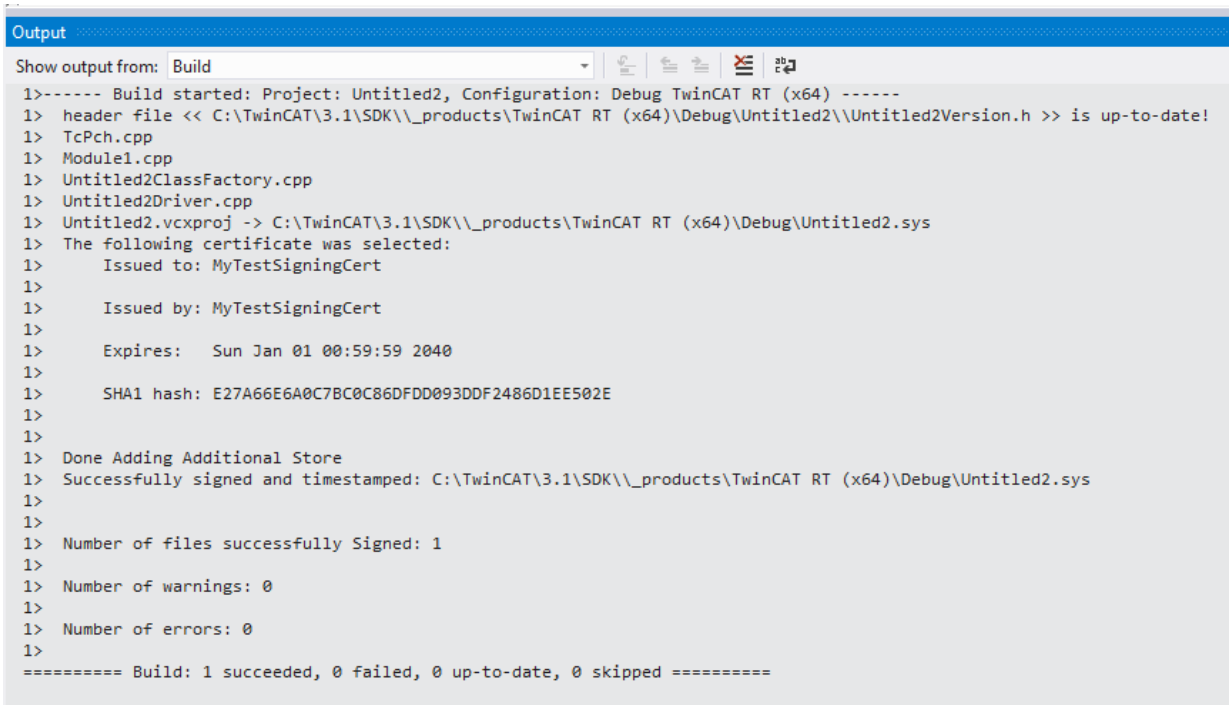
3. Auf XAR (und XAE, wenn lokaler Test)
   Signiermodus aktivieren, damit Windows die selbstsignierten Zertifikate akzeptieren kann. Das ist auf jedem System möglich, das die Module starten kann - Engineering System oder XAR (Laufzeit-)

Systeme.
Mittels Eingabeaufforderung folgendes ausführen:
`bcdedit /set testsigning yes`
und Zielsystem neu starten.

⇨ Wenn der Testsigniermodus aktiviert ist, wird dies rechts unten auf dem Desktop angezeigt. Der PC akzeptiert nun alle signierten Treiber zur Ausführung.

4. Prüfen Sie, ob eine Konfiguration mit einem in einen TwinCAT C++ Treiber implementierten TwinCAT-Modul auf dem Zielsystem aktiviert und gestartet werden kann.

⇨ Die Kompilierung des x64-Treibers generiert eine Ausgabe wie folgt:



Referenzen:
MSDN, Testzertifikate (Windows Treiber)
MSDN, MakeCert Testzertifikate (Windows Treiber),

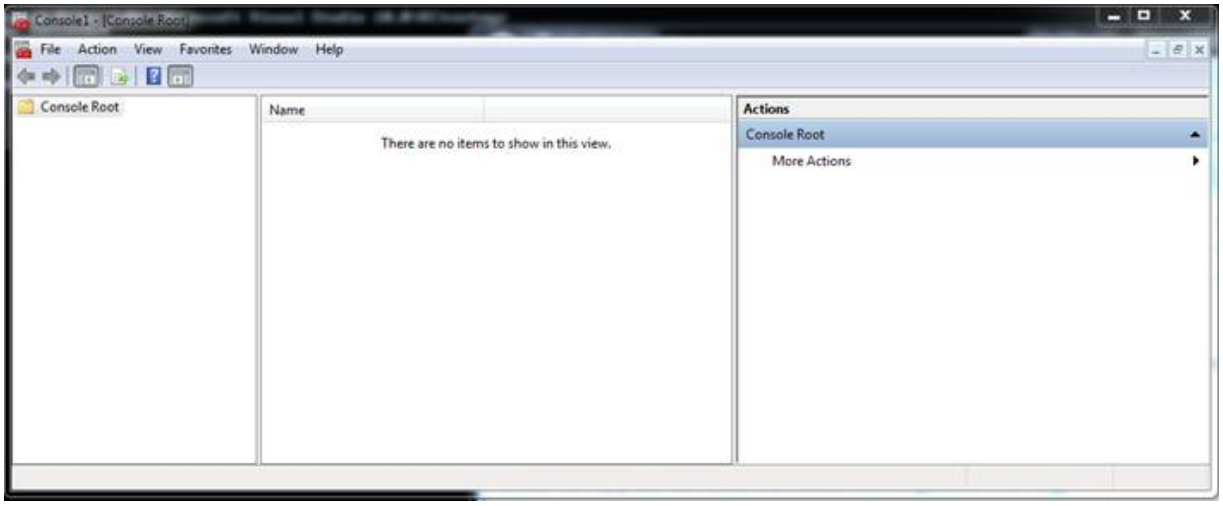
5.4.3 Testzertifikat löschen

Dieser Artikel beschreibt, wie ein Testzertifikat gelöscht wird.

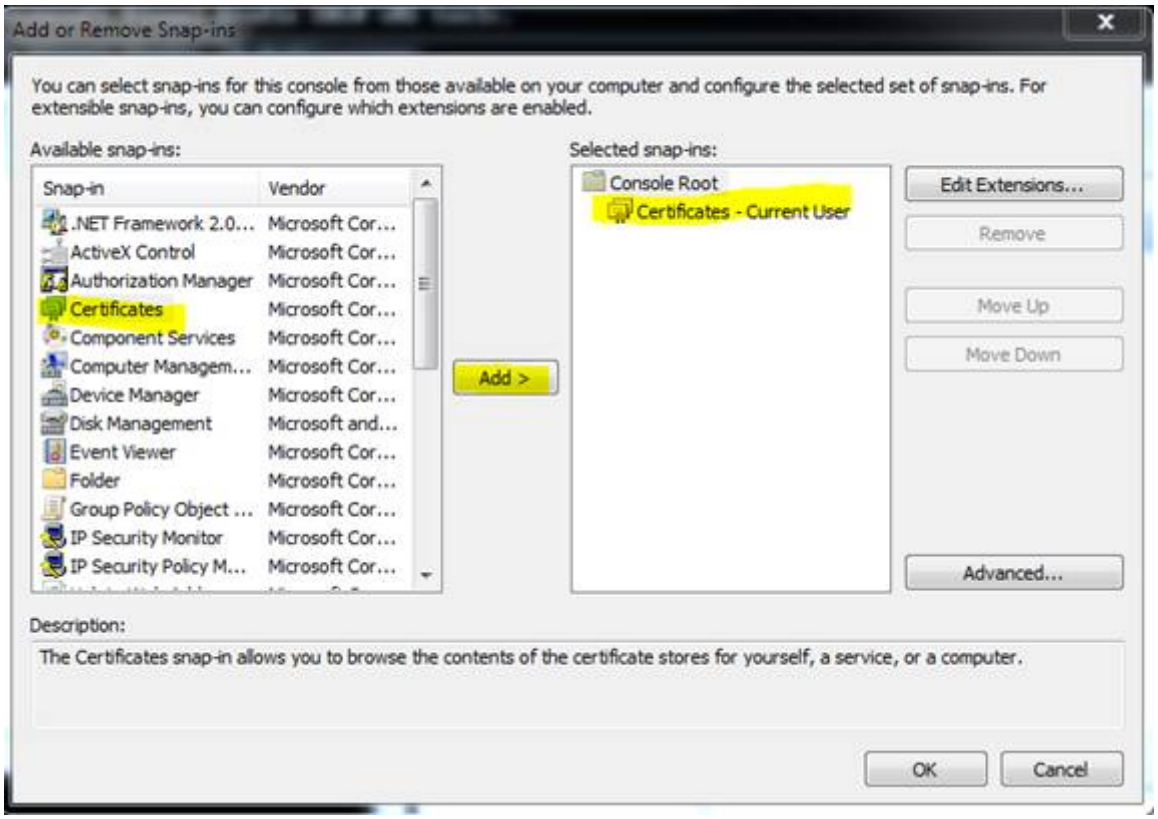
Übersicht

Ein Zertifikat kann mit der Microsoft Management Console gelöscht werden:

1. Management console MMC.exe über Start-Menü oder Oberfläche starten

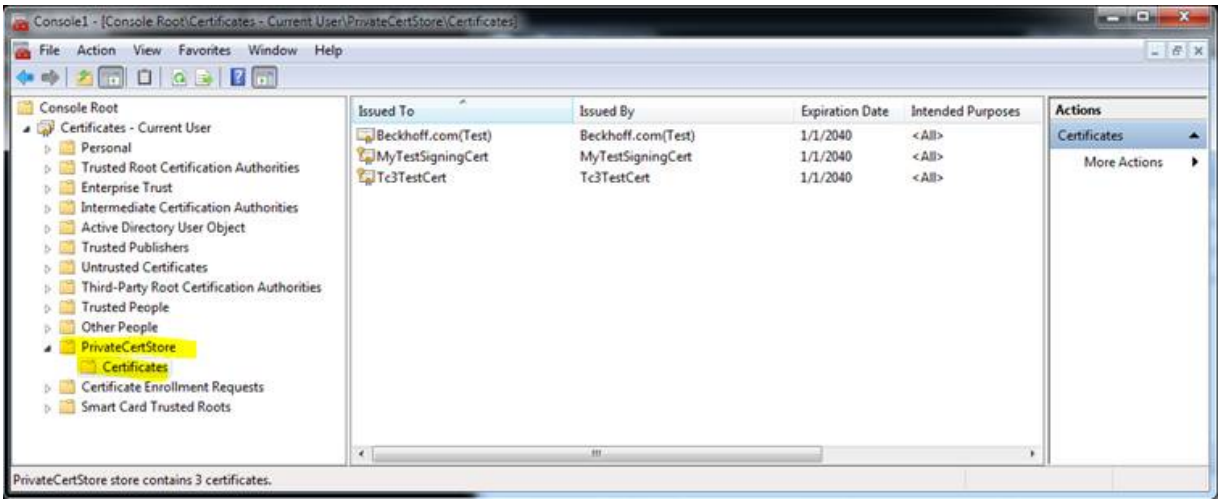


2. Im Menü auf „File“ -> „Add/Remove Snap-in..“ klicken und Zertifikat-Snap-In für den aktuellen Nutzer wählen - mit „OK“ abschließen


Add or Remove Snap-ins
You can select snap-ins for this console from those available on your computer and configure the selected set of snap-ins. For extensible snap-ins, you can configure which extensions are enabled.
Available snap-ins:
Snap-in
Vendor
.NET Framework 2.0... Microsoft Cor...
ActiveX Control Microsoft Cor...
Authorization Manager Microsoft Cor...
Certificates Microsoft Cor...
Component Services Microsoft Cor...
Computer Managem... Microsoft Cor...
Device Manager Microsoft Cor...
Disk Management Microsoft and...
Event Viewer Microsoft Cor...
Folder Microsoft Cor...
Group Policy Object ... Microsoft Cor...
IP Security Monitor Microsoft Cor...
IP Security Policy M... Microsoft Cor...
Add >
Selected snap-ins:
Console Root
Certificates - Current User
Edit Extensions...
Remove
Move Up
Move Down
Advanced...
Description:
The Certificates snap-in allows you to browse the contents of the certificate stores for yourself, a service, or a computer.
OK
Cancel

⇨ Die Zertifikate sind im Knoten unter „PrivateCertStore/Certificates“ aufgelistet.

3. Zu löschendes Zertifikat auswählen.

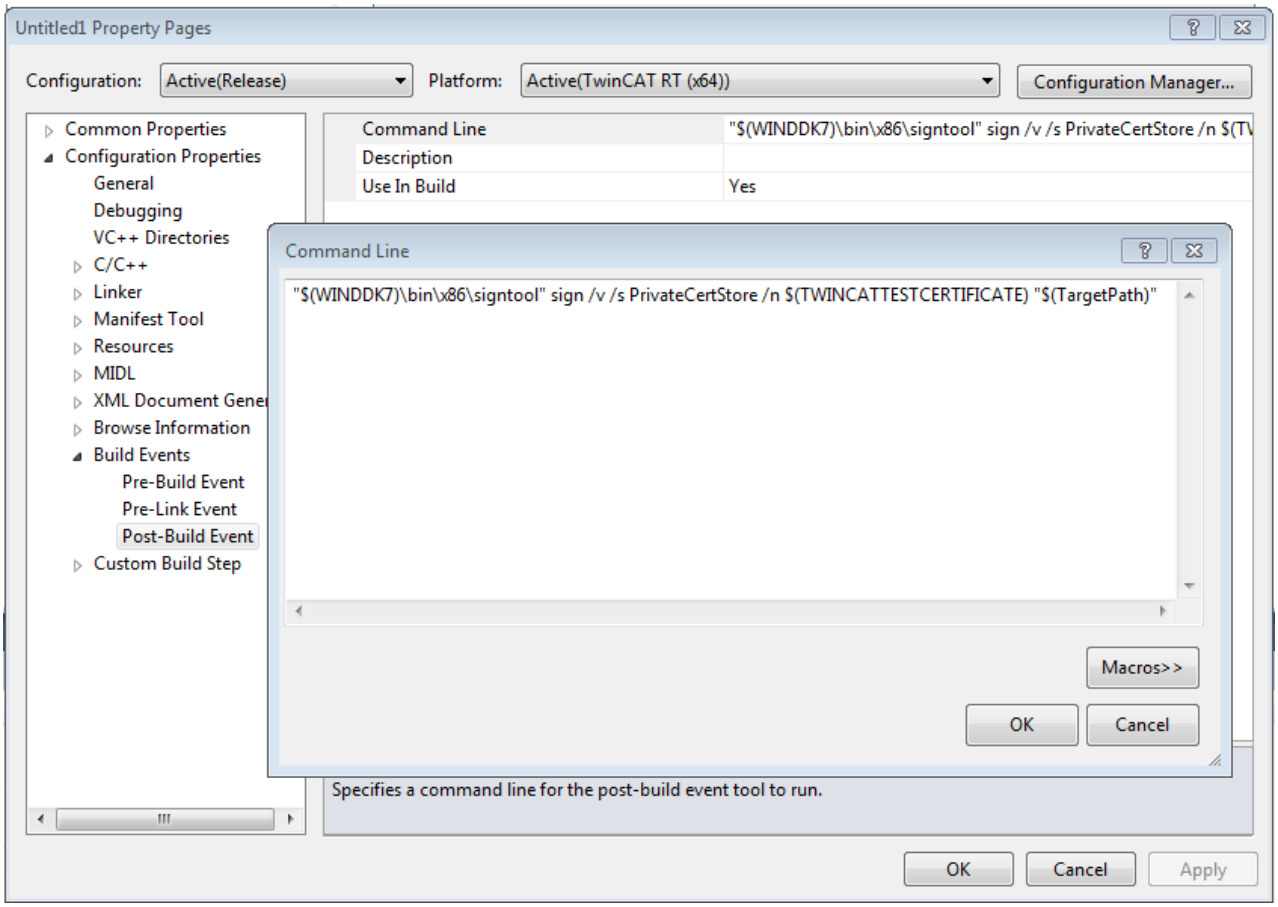


5.4.4 Kundenzertifikate

Wenn der TwinCAT C++ Klassenassistent genutzt wird, wird das Projekt unter Verwendung des zuvor beschriebenen Testzertifikatsverfahren auf x64-Zielen vorbereitet.

Dieses Testsigniersystem kann für den gesamten Engineering- und Testprozess verwendet werden.

Wenn man eine Infrastruktur erstellen und die Kernel-Treiber mit offiziellen „Microsoft trusted“-Zertifikaten unterzeichnen möchte, bieten die Postbuild-Prozeduren der Projekteigenschaften den Einsprungpunkt.



Der Kunde kann einfach den Wert der Umgebungsvariablen **TWINCATTESTCERTIFICATE** ersetzen oder ein anderes zu verwendendes Zertifikat bestimmen.
Der Kunde kann auch das ganze Signierverfahren mit dem Signiertool ändern.

6 Module

Das TwinCAT-Modulkonzept ist eines der Kernelemente für die Modularisierung moderner Maschinen. Dieses Kapitel beschreibt das Modulkonzept und den Umgang mit Modulen.

Das Modulkonzept gilt für jedes TwinCAT-Modul, nicht nur für C++ Module. Die meisten Details betreffen aber nur das Engineering von C++ Modulen.

6.1 TwinCAT-Modul

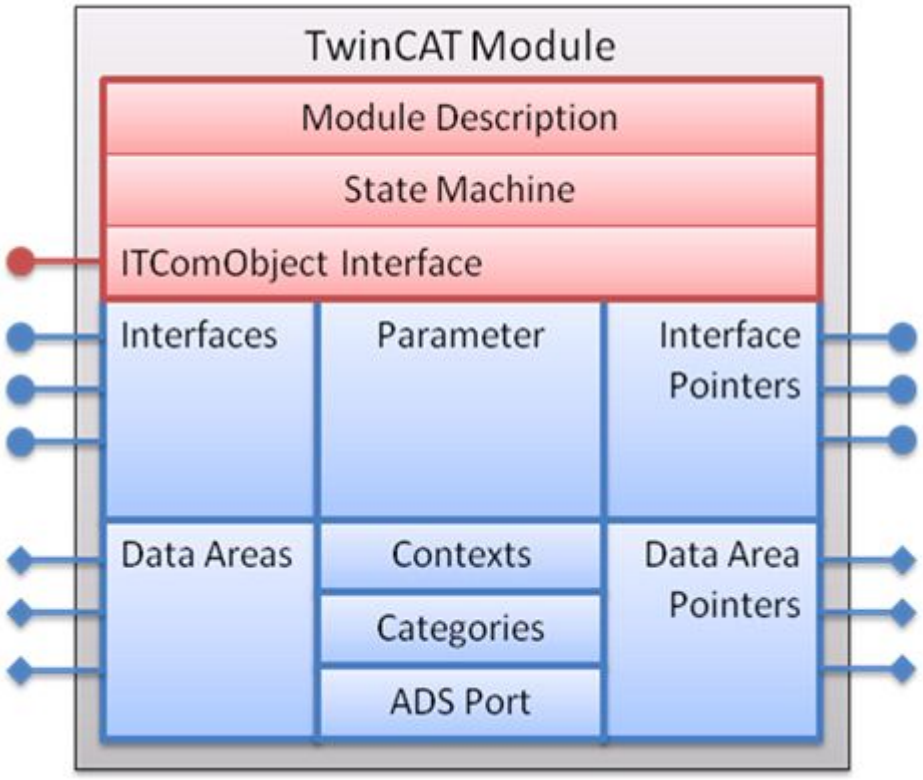
Ein TwinCAT-Modul hat eine Reihe formal definierter Eigenschaften, teilweise vorgeschrieben und teilweise optional. Die Eigenschaften sind so weit formalisiert, dass eine allgemeine Verwendung untereinander und auch von außerhalb möglich ist. Jedes Modul hat eine Modulbeschreibungsdatei (XML Format), die die Eigenschaften des Moduls beschreibt, für eine Konfiguration der Module und deren Beziehungen untereinander.

Wenn ein Modul in der TwinCAT Runtime instanziiert wird, dann meldet es sich selber bei einer zentralen Systeminstanz, dem Modulmanager, an. Dadurch wird es für andere Module und auch für allgemeine Tools erreichbar und parametrierbar. Module können unabhängig voneinander kompiliert und demzufolge auch entwickelt, getestet und aktualisiert werden. Module können sehr einfach konzipiert sein, z.B. nur eine kleine Funktion wie einen Tiefpassfilter enthalten. Sie können aber auch intern sehr komplex sein und z.B. das gesamte Steuerungssystem einer Maschinenunterbaugruppe beinhalten.

Es gibt sehr viele Anwendungen für Module; alle Aufgaben eines Automatisierungssystems können und werden in Module gepackt. Demzufolge wird nicht unterschieden, ob das Modul primär die Grundfunktionen eines Automatisierungssystems darstellt, wie Echtzeit-Tasks, Feldbus-Treiber oder ein SPS-Laufzeitsystem, oder eher benutzer- und anwendungsspezifische Algorithmen für die Steuerung oder Regelung einer Maschineneinheit.

Eigenschaften der Module

Die Abbildung unten zeigt ein gewöhnliches TwinCAT-Modul mit seinen wichtigsten Eigenschaften. Die roten Blöcke definieren vorgeschriebene, die blauen Blöcke optionale Eigenschaften.



Modulbeschreibung

Jedes TwinCATModul hat einige allgemeine Beschreibungsparameter. Dazu gehört eine ClassID, eine eindeutige Zahl, die die Klasse des Moduls eindeutig beschreibt. Jede Instanz eines Moduls hat eine Objekt-ID, die in der TwinCAT Runtime eindeutig ist. Darüber hinaus gibt es eine Parent-Objekt-ID, die auf einen möglichen logischen Parent verweist. Modulbeschreibung, Zustandsmaschine und Parameter des unten beschriebenen Moduls können über die spezifizierte und standardisierte ITComObject-Schnittstelle erreicht werden. (Siehe „Schnittstellen“).

Zustandsmaschine

Jedes Modul enthält eine Zustandsmaschine, die den Initialisierungszustand des Moduls und die Mittel, mit denen dieser Zustand von außen verändert werden kann, beschreibt. Diese Zustandsmaschine beschreibt nur den allgemeinen Zustand des Moduls und definiert den Ablauf von Modulgenerierung, -parametrierung und Herstellung der Verbindung mit den anderen Modulen.

Anwendungsspezifische Zustände (z.B. von Feldbus oder Treiber) können in ihren eigenen Zustandsmaschinen stromabwärts beschrieben werden. Die Zustandsmaschine der TwinCAT-Module definiert die Zustände INIT, PREOP, SAFEOP und OP. Auch wenn es dieselben Zustandsbezeichnungen sind wie unter EtherCAT-Feldbus sind es doch nicht die gleichen Zustände.

Wenn das TwinCAT-Modul einen Feldbustreiber für EtherCAT implementiert, hat es zwei Zustandsmaschinen (Modul- und Feldbuszustandsmaschine), die nacheinander durchlaufen werden. Die Modulzustandsmaschine muss den Betriebszustand (OP) erreicht haben, bevor die Feldbusmaschine überhaupt starten kann.

Parameter

Module können Parameter haben, die während der Initialisierung oder später während der Laufzeit (OP-Zustand) gelesen oder geschrieben werden können. Jeder Parameter ist durch eine Parameter-ID gekennzeichnet. Die Eindeutigkeit der Parameter-ID kann global, eingeschränkt global oder modulspezifisch sein. Mehr hierzu im Abschnitt „ID Management“. Neben der Parameter-ID enthält der Parameter die aktuellen Daten; der Datentyp hängt vom Parameter ab und ist für die jeweilige Parameter-ID eindeutig definiert.

Schnittstellen

Schnittstellen bestehen aus einem definierten Satz an Methoden (Funktionen), die Module anbieten und über die sie z.B. von anderen Modulen kontaktiert werden können. Schnittstellen sind ähnlich wie Parameter durch eine eindeutige ID charakterisiert. Ein Modul muss mindestens die ITComObject-Schnittstelle unterstützen, kann aber daneben so viele Schnittstellen wie gewünscht beinhalten. Der Nutzer einer Schnittstelle kann über die Schnittstelle durch Aufruf eines „Query Interface“ mit Angabe der entsprechenden Schnittstellen-ID abfragen, ob ein Modul weitere Schnittstellen unterstützt. Weitere Erläuterungen befinden sich im Abschnitt „TcCOM“.

Schnittstellenzeiger

Schnittstellenzeiger verhalten sich wie das Gegenstück von Schnittstellen. Wenn ein Modul eine Schnittstelle eines anderen Moduls nutzen möchte, muss es einen Schnittstellenzeiger des entsprechenden Schnittstellentyps haben und dafür sorgen, dass dieser auf das andere Modul zeigt. Danach können die Methoden des anderen Moduls verwendet werden.

Schnittstellenzeiger werden normalerweise beim Start der Zustandsmaschine gesetzt. Beim Übergang von INIT zu PREOP (IP) empfängt das Modul die Objekt-ID des anderen Moduls mit der entsprechenden Schnittstelle, beim Übergang PREOP zu SAFEOP (PS) oder SAFEOP zu OP (SO) wird die Instanz des anderen Moduls mit dem Modulmanager durchsucht und die entsprechende Schnittstelle mit der Methode Query Interface gesetzt. Beim Zustandsübergang in die entgegengesetzte Richtung, von SAFEOP zu PREOP (SP) oder OP zu SAFEOP (OS) muss die Schnittstelle wieder freigegeben werden.

Datenbereiche

Module können Datenbereiche enthalten, die von der Umgebung verwendet werden können (z.B. von anderen Modulen). Diese Datenbereiche können beliebige Daten enthalten. Sie werden oft für Prozessabbilddaten (Ein- und Ausgänge) genutzt. Die Struktur der Datenbereiche wird in der Gerätebeschreibung des Moduls definiert. Wenn ein Modul Datenbereiche hat, die es für andere zugänglich machen möchte, implementiert es die ITcADI-Schnittstelle, die den Zugriff auf die Daten ermöglicht. Datenbereiche können Symbolinformationen enthalten, die die Struktur des jeweiligen Datenbereichs im Einzelnen beschreiben.

Datenbereichszeiger

Wenn ein Modul auf den Datenbereich anderer Module zugreifen möchte, kann es Datenbereichszeiger enthalten. Diese werden normalerweise während der Initialisierung der Zustandsmaschine auf Datenbereiche oder Datenbereichsabschnitte anderer Module gesetzt.

Kontext

In diesem Zusammenhang ist Kontext als Zeitkontext zu verstehen. Kontexte werden u.a. für die Konfiguration der Module benötigt. Einfache Module arbeiten normalerweise in einem einzigen Zeitkontext, demzufolge muss er nicht näher spezifiziert werden. Andere Module können teilweise in mehreren Kontexten aktiv sein (z.B. ein EtherCAT Master kann mehrere zeitunabhängige Tasks unterstützten, oder eine Regelschleife kann Regelschleifen der darunterliegenden Ebene in anderer Zykluszeit abarbeiten). Wenn ein Modul mehr als einen zeitabhängigen Kontext hat, muss das in der Modulbeschreibung angegeben werden.

Kategorien

Module können Kategorien unterstützen, die dem Konfigurator die Verbindung verschiedener Module untereinander erleichtern. Eine Kategorie wird über eine Kategorie-ID definiert, die ähnlich wie Parameter- oder Schnittstellen-ID definiert ist (siehe ID Management). Wenn ein Schnittstellenzeiger eines Moduls mit der Schnittstelle eines anderen Moduls in Verbindung treten soll, kann in der Modulbeschreibung festgelegt werden, dass er nur mit Modulen verbunden werden kann, die eine bestimmte Kategorie unterstützen. Dadurch wird die Auswahl für den Konfigurator und demzufolge auch für den Nutzer deutlich einfacher und leichter verständlich.

ADS

Jedes Modul ist über ADS via ITComObject-Schnittstelle automatisch erreichbar. Z.B. kann auf Parameter oder Zustandsmaschine eines Moduls mit AdsRead und AdsWrite zugegriffen werden. Bei komplexeren Modulen gibt es die zusätzliche Möglichkeit der Implementierung eines ADS Ports im Modul selbst, über den der gesamte ADS-Kommunikationspfad zur Verfügung steht.

Systemmodul

Die TwinCAT Laufzeit stellt darüber hinaus eine Reihe sogenannter Systemmodule zur Verfügung, die die grundlegenden Dienste der Laufzeit für andere Module zur Verfügung stellen. Diese Systemmodule haben eine feste, konstante Objekt-ID, über die die anderen Module auf sie zugreifen können. Ein Beispiel eines solchen Systemmoduls ist das Echtzeitsystem, das die grundlegenden Dienste des Echtzeitsystems, d.h. die Generierung von Echtzeit-Tasks, via ITcRTime-Schnittstelle zur Verfügung stellt. Auch der ADS Router ist als Systemmodul implementiert, so dass andere Module an dieser Stelle ihren ADS Port anmelden können.

Erstellung von Modulen

Module können sowohl in C++ als auch in IEC 61131-3 erstellt werden. Die objektorientierten Erweiterungen des TwinCAT SYS von TwinCAT werden hierzu verwendet. Module aus den beiden Welten können über Schnittstellen auf die gleiche Weise wie reine C++ Module untereinander interagieren. Mit Hilfe der objektorientierten Erweiterung in PLC 3.0 werden die gleichen Schnittstellen bereitgestellt wie in C++.

Die SPS-Module melden sich ebenfalls selber beim Modulmanager an und sind demzufolge über ihn erreichbar. Die Komplexität von SPS-Modulen ist unterschiedlich. Es macht keinen Unterschied, ob nur ein kleines Filtermodul generiert oder ein komplettes SPS-Programm in ein Modul hineingepackt wird. Und in

der Tat ist jedes SPS-Programm aufgrund der Automation ein Modul im Sinne der TwinCAT-Module. Jedes klassische SPS-Programm wird automatisch in ein Modul gepackt und meldet sich selber beim Modulmanager und bei einem oder mehreren Task-Modulen an. Der Zugriff auf die Prozessdaten eines SPS-Moduls (z.B. Mapping in Bezug auf einen Feldbus-Treiber) wird ebenfalls über die definierten Datenbereiche und ITcADI gesteuert.

Dieses Verhalten bleibt transparent und unsichtbar für den SPS-Programmierer, so lange bis er beschließt ausdrücklich Teile des SPS-Programms als TwinCAT-Module zu definieren, damit er diese mit geeigneter Flexibilität nutzen kann.

6.2 TwinCAT Component Object Model (TcCOM)

Das TwinCAT Component Object Model definiert die Eigenschaften und das Verhalten der Module. Das vom „Component Object Model“ (COM) von Microsoft Windows abgeleitete Modell beschreibt die Art und Weise, wie verschiedene Software-Komponenten, die unabhängig voneinander entwickelt und kompiliert wurden, zusammen arbeiten können. Damit das möglich ist, müssen ein genau definierter Verhaltensmodus und die Beachtung von Schnittstellen des Moduls ein für alle Mal genau definiert werden, sodass sie mit einander verbunden werden können, selbst dann wenn sie unterschiedlicher Herkunft sind.

Auch wenn das Component Object Model aus der Windows-Welt mehr und mehr durch die .NET Herangehensweise ersetzt wird, sind die Prozeduren und Definitionen von COM für eine Echtzeitverwendung in TwinCAT viel geeigneter als die .NET Herangehensweise. Bei .NET werden viele Dinge erst während der Laufzeit untereinander geklärt (Reflektion). Das Verhalten des Garbage Collectors eignet sich ebenfalls nicht sehr gut für Echtzeit.

TcCOM macht demzufolge Anleihen bei COM, wobei lediglich eine Untermenge von COM verwendet wird. Allerdings enthält TcCOM im Vergleich zu COM zusätzliche Definitionen, die über COM hinausgehen, z.B. das Modul Zustandsmaschine.

Schnittstellen

Eine wichtige Komponente von COM, und also auch von TcCOM, sind Schnittstellen.

Schnittstellen definieren einen Satz Methoden, die zusammengefasst werden, um eine bestimmte Aufgabe zu erfüllen. Eine Schnittstelle wird mit einer eindeutigen ID bezeichnet, die nachdem sie verwendet wurde, nie mehr geändert werden darf. Dank der ID können die verschiedenen Module feststellen, ob sie mit anderen Modulen zusammenarbeiten können, selbst wenn Jahre zwischen den Entwicklungen der verschiedenen Module liegen.

TcCOM selber definiert bereits eine ganze Reihe an Schnittstellen, die in manchen Fällen vorgeschrieben (z.B. ITComObject), aber in den meisten Fällen optional sind. Viele Schnittstellen machen nur in bestimmten Anwendungsbereichen Sinn. Andere Schnittstellen sind dermaßen allgemein, dass sie häufig wiederverwendet werden können. Private Schnittstellen sind ebenfalls erlaubt, sodass z.B. zwei Module eines einzelnen Herstellers in der Lage sind, miteinander in Verbindung zu treten.

Alle Schnittstellen werden von der grundlegenden Schnittstelle ITcUnknown abgeleitet, die, wie die unbekannte Schnittstelle von COM, die grundlegenden Dienste zur Abfrage von anderen Schnittstellen des Moduls (TcQueryInterface) und zur Steuerung der Lebensdauer des Moduls (TcAddRef und TcRelease) bereitstellt.

Die ITComObject-Schnittstelle, die von jedem Modul implementiert sein muss, enthält Methoden um auf den Namen, die Objekt-ID, die Objekt-ID des Parent, die Parameter und die Zustandsmaschine des Moduls zuzugreifen. Zu den allgemeinen Schnittstellen, die von vielen Modulen verwendet werden, gehören: ITcCyclic für zyklischen Aufruf eines Moduls – entsprechend der ITcCyclicCaller-Schnittstelle, mit Hilfe derer Module sich bei einem anderen anmelden können, um einen zyklischen Aufruf zu erhalten.

Für Prozessdaten stehen die vergleichbaren Schnittstellen TcIoCyclic und ITcIoCyclicCaller zur Verfügung, um Ein- und Ausgänge zu aktualisieren. Mit Hilfe der ITcADI-Schnittstelle kann auf die Datenbereiche eines Moduls zugegriffen werden. Die ITcTask-Schnittstelle, die bei Modulen implementiert ist, die einen Task eines Echtzeitsystems darstellen, stellt Informationen bezüglich der Zykluszeit, der Priorität und anderer Informationen zum Task zur Verfügung.

Der Modulmanager implementiert eine weitere wichtige Schnittstelle, ITcComObjectServer, die von allen Modulen im Verlauf der Initialisierung referenziert wird. Mit Hilfe dieser Schnittstelle kann nach anderen Modulen über deren Objekt-ID gesucht werden, und das dem Manager gehörende Modul beim Modulmanager an- und abgemeldet werden. Über diese Schnittstelle können auch neue Module generiert werden (vergleiche mit „Class Factories“).

Es wurden bereits eine ganze Reihe allgemeiner Schnittstellen definiert. Allgemeine Schnittstellen haben den Vorteil, dass deren Verwendung den Austausch und die Wiederverwertung von Modulen unterstützt. Nur dann wenn keine geeigneten allgemeinen Schnittstellen bestehen, sollten private Schnittstellen definiert werden.

Class Factories

Für die Erstellung von in C++ entwickelten Modulen werden sogenannte „Class Factories“ verwendet. Alle Module, die in einem Compiler-Kontext erstellt wurden, besitzen eine gemeinsame Class Factory. Die Class Factory meldet sich einmal beim Modulmanager an und bietet ihre Dienste für die Erstellung bestimmter Modulklassen an. Die Modulklassen sind durch die eindeutige Klassen-ID des Moduls gekennzeichnet. Wenn der Modulmanager ein neues Modul anfordert (auf Grundlage der Initialisierungsdaten des Konfigurators oder durch andere Module während der Laufzeit), wählt das Modul die richtige Class Factory auf Grundlage der Klassen-ID aus und veranlasst die Erzeugung des Moduls über seine ITcClassFactory-Schnittstelle.

Modul-Lebensdauer

Auf ähnliche Weise wie im Falle von COM wird die Lebensdauer eines Moduls über einen Verweiszähler bestimmt. Jedes Mal, wenn eine Schnittstelle eines Moduls abgefragt wird, wird der Verweiszähler inkrementiert. Er wird wieder dekrementiert, wenn die Schnittstelle freigegeben wird. Eine Schnittstelle wird auch bei der Anmeldung eines Moduls beim Modulmanager abgefragt, (die ITComObject-Schnittstelle), sodass der Verweiszähler zumindest auf eins steht. Bei der Abmeldung wird er erneut dekrementiert. Wenn der Zähler den Wert 0 erreicht, löscht das Modul sich selbst automatisch - normalerweise nach der Abmeldung beim Modulmanager. Wenn aber bereits ein anderes Modul einen Verweis hält (einen Schnittstellenzeiger besitzt), dann besteht das Modul weiter - und der Schnittstellenzeiger bleibt so lange gültig, bis dieser Zeiger auch freigegeben wird.

ID Management

Es wird eine ganze Reihe von IDs für die Interaktion der Module untereinander verwendet. Diese müssen alle eindeutig sein. Für COM wird demzufolge eine 128 Bit lange Zufallszahl (GUID) für solche IDs verwendet, mit der begründeten Annahme, dass aufgrund des großen zulässigen Zahlenfelds sich zwei identische GUIDs einfach nicht ergeben werden. Unglücklicherweise ist die Handhabung solch langer Zahlen auch sehr kompliziert, selbst für moderne CPUs, und ist insbesondere in der Echtzeitumgebung ungünstig. Aus diesem Grund verwendet TcCOM eine Kombination aus GUIDs und 32 Bit langen Zahlen, die deutlich effizienter gehandhabt werden können, aber es kann nicht länger mit Zufallszahlen gearbeitet werden.

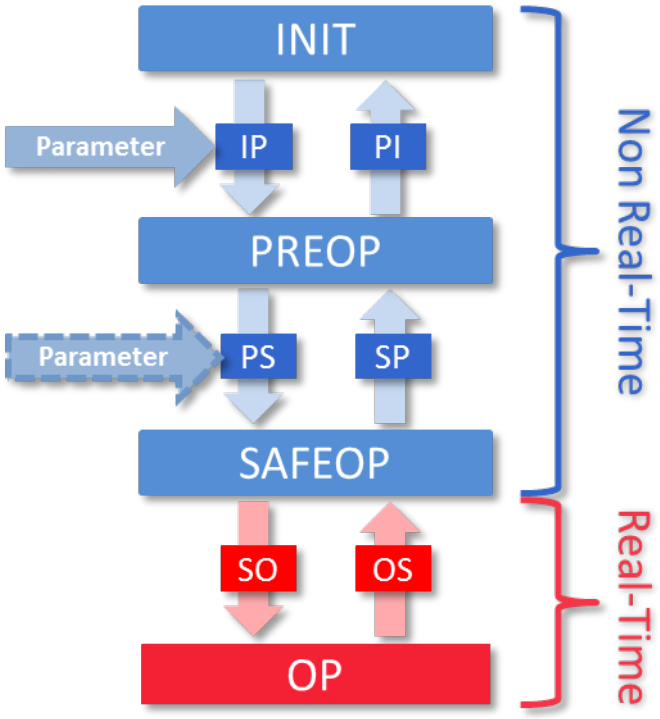
Um sicherzustellen, dass die Zahlen eindeutig sind, müssen sie verwaltet werden. Die Aufteilung in einen zentral verwalteten (die oberen 16 Bits der Zahl) und einen lokal verwalteten Bereich (untere 16 Bits) ermöglichen eine flexible Handhabung der ID-Nummern. Jedem Entwickler oder Hersteller wird eine oder mehrere 16 Bit Nummer(n) zugewiesen, die als HiWord für seine eigenen Parameter-IDs, Objekt-IDs, usw. verwendet werden.

Die unteren 16 Bit liegen innerhalb deren eigenen Verwaltungssphäre und müssen nur innerhalb deren eigenen Bereiche eindeutig sein. TcCom verwendet eindeutige GUIDs für class IDs, InterfaceIDs und categoryIDs, da diese lediglich im Verlauf der Initialisierung verwendet werden, sodass der höhere CPU-Aufwand sich nicht auf den Echtzeitkontext auswirkt.

6.3 Zustandsmaschine

Neben den Zuständen (INIT, PREOP, SAFEOP und OP) gibt es entsprechende Zustandsübergänge, innerhalb derer allgemeine oder modulspezifische Aktionen auszuführen sind oder ausgeführt werden können. Die Zustandsmaschine ist sehr einfach konzipiert; in jedem Falle gibt es nur Übergänge zum nächsten oder vorherigen Schritt.

Hieraus ergeben sich die Zustandsübergänge: INIT zu PREOP (IP), PREOP zu SAFEOP (PS) und SAFEOP zu OP (SO). In umgekehrter Richtung bestehen die folgenden Zustandsübergänge: OP zu SAFEOP (OS), SAFEOP zu PREOP (SP) und PREOP zu INIT (PI). Einschließlich bis zum SAFEOP-Zustand finden alle Zustände und Zustandsübergänge innerhalb des Nicht-Echtzeitkontextes statt. Nur der Übergang SAFEOP zu OP, der Zustand OP und der Übergang OP zu SAFEOP finden im Echtzeitkontext statt. Diese Differenzierung hat einen Einfluss, wenn Ressourcen alloziert oder freigegeben werden, oder wenn Module sich bei anderen Modulen an- oder abmelden.



INIT

Der INIT-Zustand ist nur ein virtueller Zustand. Sofort nach Erstellung eines Moduls wechselt das Modul von INIT zu PREOP, d.h. der IP-Zustandsübergang wird ausgeführt. Die Instanziierung und der IP-Zustandsübergang erfolgen immer zusammen, so dass das Modul nie im INIT-Zustand bleibt. Nur beim Entfernen des Moduls, bleibt es kurzzeitig im INIT-Zustand.

INIT zu PREOP (IP)

Während des IP-Zustandsübergangs meldet sich das Modul mit seiner eindeutigen Objekt-ID beim Modulmanager an. Die Initialisierungsparameter, die auch während der Objekterstellung alloziert werden, werden an das Modul weitergegeben. Bei diesem Übergang kann das Modul keine Verbindung zu anderen Modulen herstellen, weil nicht sicher ist, ob die anderen Module bereits bestehen und beim Modulmanager angemeldet sind. Wenn das Modul Systemressourcen benötigt (z.B. Speicherplatz), können diese während des Zustandsübergangs alloziert werden. Alle allozierten Ressourcen müssen dann entsprechend beim Übergang PREOP zu INIT (PI) wieder freigegeben werden.

PREOP

Im PREOP-Zustand ist das Modul vollständig erstellt und auch normalerweise vollständig parametriert, auch wenn möglicherweise beim Übergang von PREOP zu SAFEOP weitere Parameter hinzukommen. Das Modul ist im Modulmanager angemeldet, es wurden aber noch keine Verbindungen mit anderen Modulen herstellt.

PREOP zu SAFEOP (PS)

In diesem Zustandsübergang kann das Modul Verbindungen mit anderen Modulen herstellen. Zu diesem Zweck hat es normalerweise, neben anderen Dingen, Objekt-IDs von anderen Modulen mit den Initialisierungsdaten erhalten, die nun über den Modulmanager in reale Verbindungen mit diesen Modulen umgewandelt werden.

Der Übergang kann sowohl im Allgemeinen vom System, gemäß dem Konfigurator, als auch von einem anderen Modul (z.B. dem Parent-Modul) veranlasst werden. Im Verlauf dieses Zustandsübergangs können auch weitere Parameter übergeben werden. So kann z.B. das Parent-Modul eigene Parameter an das Child-Modul übergeben.

SAFEOP

Das Modul ist noch im Nicht-Echtzeitkontext und wartet darauf, vom System oder von anderen Modulen in den OP-Zustand geschaltet zu werden.

SAFEOP zu OP (SO)

Sowohl der Zustandsübergang von SAFEOP zu OP, als auch der Zustand OP, als auch der Übergang von OP zu SAFEOP finden im Echtzeitkontext statt! Ressourcen des Systems dürfen nicht mehr alloziert werden. Auf der anderen Seite können nun Ressourcen von anderen Modulen angefordert werden und Module können sich bei anderen Modulen anmelden, z.B. im Verlauf von Tasks, um einen zyklischen Aufruf zu erhalten.

OP

Im OP-Zustand nimmt das Modul seine Arbeit auf und ist im Sinne des TwinCAT-Systems voll aktiv. Alle Zustandsmaschinen stromabwärts können nun starten.

OP zu SAFEOP (OS)

Dieser Zustandsübergang findet im Echtzeitkontext statt. Alle Aktionen aus dem SO-Übergang werden umgekehrt und alle beim SO-Übergang angeforderten Ressourcen werden wieder freigegeben.

SAFEOP zu PREOP (SP)

Alle Aktionen vom PS-Übergang werden umgekehrt und alle beim PS-Übergang angeforderten Ressourcen werden wieder freigegeben.

PREOP zu INIT (PI)

Alle Aktionen vom IP-Übergang werden umgekehrt und alle beim IP-Übergang angeforderten Ressourcen werden wieder freigegeben. Das Modul meldet sich beim Modulmanager ab und löscht sich normalerweise selbst (siehe „Lebensdauer“).

6.4 Klassenbeschreibungsdateien

Die Klassen der Module werden in den Klassenbeschreibungsdateien (TwinCAT Module Class; *.tmc) beschrieben. In dieser Datei beschreibt der Modulentwickler Eigenschaften und Schnittstellen des Moduls, so dass andere es nutzen und einbetten können. Neben den allgemeinen Informationen (Herstellerangaben, Klassen-ID des Moduls, usw.) werden besondere und manchmal optionale Eigenschaften des Moduls beschrieben.

- Vom Modul unterstützte Kategorien
- Vom Modul implementierte Schnittstellen
- Datenbereiche im Modul
- Für das Modul festzulegende Parameter
- Schnittstellenzeiger, die gesetzt werden können

- Datenzeiger, die gesetzt werden können

Die Klassenbeschreibungsdateien werden vom Konfigurator des Systems in erster Linie als Grundlage für die Einbindung einer Instanz des Moduls in die Konfiguration, zum Festlegen der Parameter und zwecks Konfiguration der Verbindungen mit anderen Modulen verwendet. Sie enthalten zudem die Beschreibung aller Datentypen in den Modulen, die dann vom Konfigurator in sein allgemeines Datentypsystem aufgenommen werden. Auch komplexere Konfigurationen mehrerer Module können in den Klassenbeschreibungsdateien beschrieben werden, die, wie bereits in der Vergangenheit, für eine spezifische Anwendung vorkonfiguriert und verbunden sind. Demzufolge kann ein Modul für eine komplexe Maschineneinheit, die intern aus einer Reihe von Untermodulen besteht, bereits im Verlauf der Entwicklungsphase als ein Ganzes definiert und vorkonfiguriert werden.

Instanzenbeschreibungsdateien

Ein Instanz eines bestimmten Moduls wird in der Instanzenbeschreibungsdatei (TwinCAT Module Instance; *.tmi) beschrieben. Die Instanzenbeschreibungen sind ähnlich aufgebaut wie die Klassenbeschreibungen, aber sie enthalten bereits konkrete Festlegungen der Parameter, Schnittstellenzeiger usw. für die besondere Instanz des Moduls innerhalb eines Projekts.

Die Instanzenbeschreibungsdateien werden normalerweise vom Konfigurator erstellt, wenn eine Instanz einer Klassenbeschreibung für ein konkretes Projekt erstellt wird. Sie dienen hauptsächlich dem Datenaustausch zwischen allen an der Konfiguration beteiligten Tools. Allerdings können die Instanzenbeschreibungen auch projektübergreifend genutzt werden, wenn z.B. ein besonders parametriertes Modul in einem neuen Projekt erneut verwendet werden soll.

7 Module - Handhabung

TcCOM Module werden definiert und implementiert. Anschließend können sie

- ausgetauscht werden: Module exportieren [▶ 31], Module importieren [▶ 32]
- gestartet werden

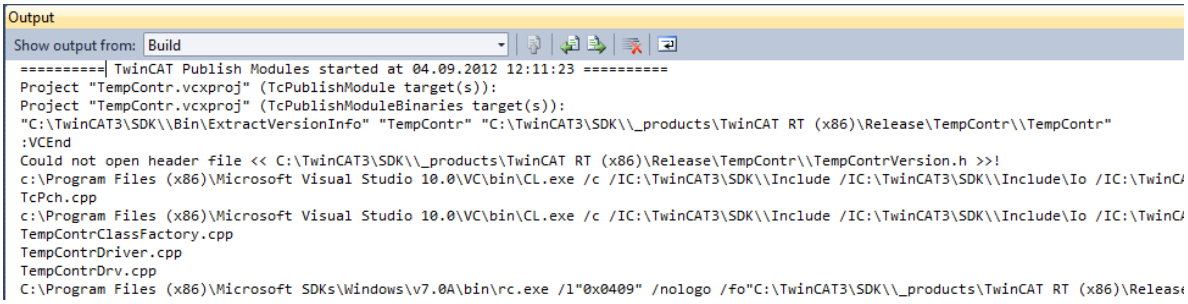
In diesem Abschnitt wird der Umgang mit Modulen beschrieben.

7.1 Module exportieren

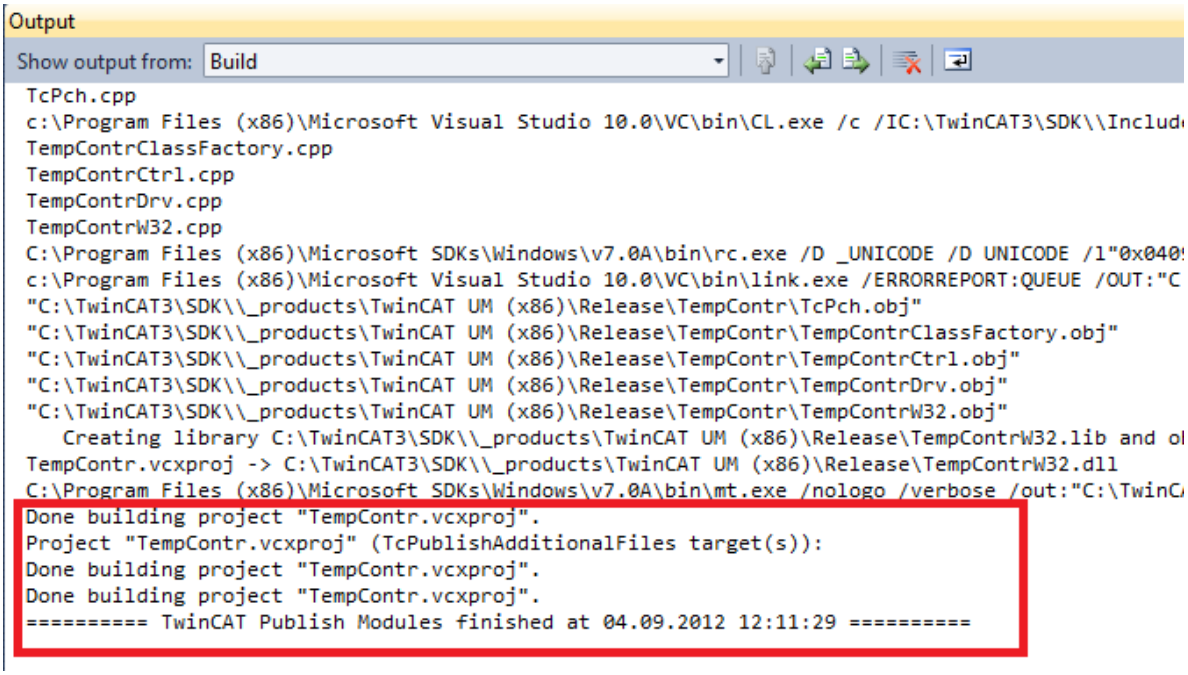
Dieser Artikel beschreibt, wie ein TwinCAT 3 Modul, das auf jedem anderen TwinCAT PC laufen kann, exportiert wird.

Folgende Schritte sind zu erledigen

1. Implementierung eines TwinCAT 3 C++ Projekts auf einem mit Visual Studio-Version ausgestatteten Engineering PC. Siehe Schnellstartbeispiel [▶ 37] „Ein TwinCAT 3 Projekt erstellen“, die TwinCAT-Module wie beschrieben implementieren, kompilieren und das Modul vor dem Export testen.
2. Da das Ergebnismodul in der Lage sein sollte, auf jeder Maschine verwendet zu werden, wird TwinCAT eine 32-Bit- und eine 64-Bit-Version des Moduls generieren.
   Da x64bit-Module unterzeichnet sein müssen, muss auf der Maschine, die das Modul exportiert, ein Zertifikat installiert sein. Siehe x64: Treibersignierung [▶ 16], wie ein Zertifikat zu generieren und installieren ist.
   (Schritt 3 kann auf einem Engineering oder 32-Bit-System ausgelassen werden)
3. Um ein TC 3 C++ Modul zu exportieren, klicken Sie einfach mit der rechten Maustaste auf das Modulprojekt im Solution-Baum und wählen „TwinCAT Publish Modules“ aus.
   ⇨ Daraufhin wird das Modul kompiliert (Rebuild) - der erfolgreiche Export wird im Ausgabefenster „Build“ angezeigt.

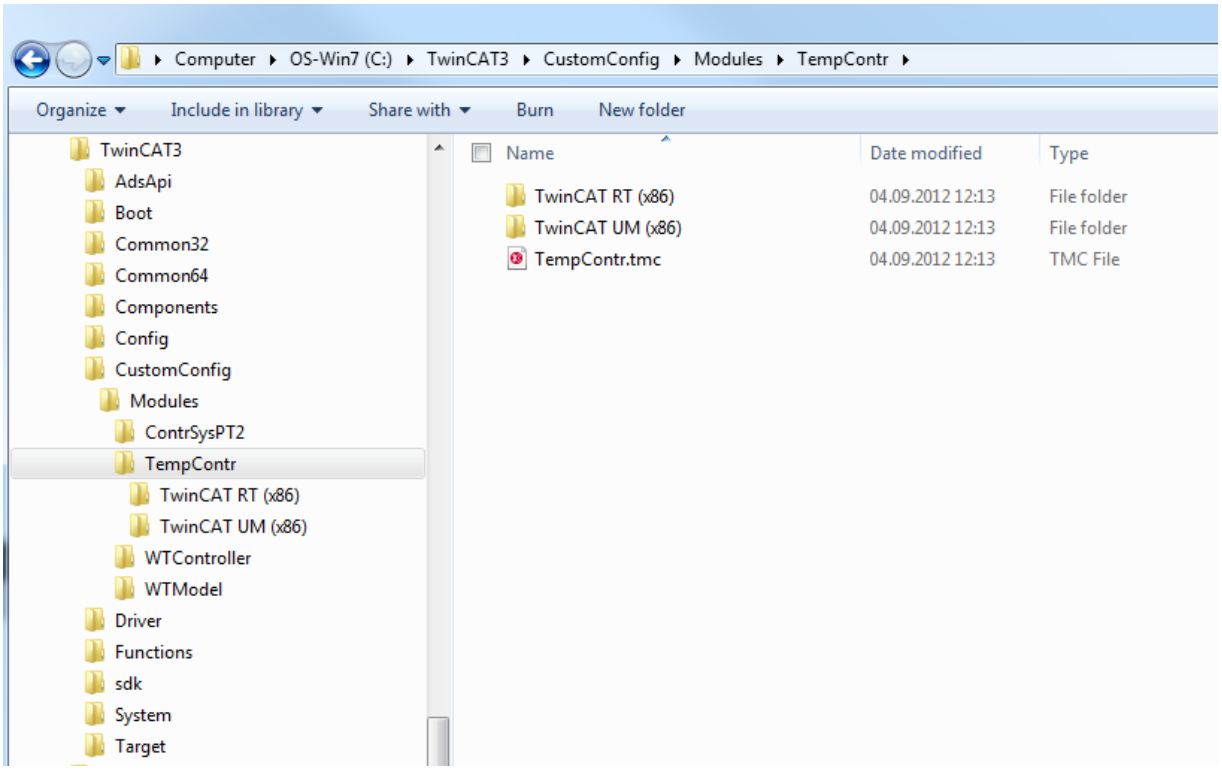

Output
Show output from: Build
========== TwinCAT Publish Modules started at 04.09.2012 12:11:23 ==========
Project "TempContr.vcxproj" (TcPublishModule target(s)):
Project "TempContr.vcxproj" (TcPublishModuleBinaries target(s)):
"C:\TwinCAT3\SDK\\Bin\ExtractVersionInfo" "TempContr" "C:\TwinCAT3\SDK_products\TwinCAT RT (x86)\Release\TempContr\\TempContr"
:VCEnd
Could not open header file << C:\TwinCAT3\SDK_products\TwinCAT RT (x86)\Release\TempContr\\TempContrVersion.h >>!
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\CL.exe /c /IC:\TwinCAT3\SDK\\Include /IC:\TwinCAT3\SDK\\Include\Io /IC:\TwinCA
TcPch.cpp
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\CL.exe /c /IC:\TwinCAT3\SDK\\Include /IC:\TwinCAT3\SDK\\Include\Io /IC:\TwinCA
TempContrClassFactory.cpp
TempContrDriver.cpp
TempContrDrv.cpp
C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\rc.exe /l"0x0409" /nologo /fo"C:\TwinCAT3\SDK_products\TwinCAT RT (x86)\Release

Most important is the success report at the end:


Output
Show output from: Build
TcPch.cpp
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\CL.exe /c /IC:\TwinCAT3\SDK\\Includ
TempContrClassFactory.cpp
TempContrCtrl.cpp
TempContrDrv.cpp
TempContrW32.cpp
C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\rc.exe /D _UNICODE /D UNICODE /l"0x040
c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\link.exe /ERRORREPORT:QUEUE /OUT:"C
"C:\TwinCAT3\SDK_products\TwinCAT UM (x86)\Release\TempContr\TcPch.obj"
"C:\TwinCAT3\SDK_products\TwinCAT UM (x86)\Release\TempContr\TempContrClassFactory.obj"
"C:\TwinCAT3\SDK_products\TwinCAT UM (x86)\Release\TempContr\TempContrCtrl.obj"
"C:\TwinCAT3\SDK_products\TwinCAT UM (x86)\Release\TempContr\TempContrDrv.obj"
"C:\TwinCAT3\SDK_products\TwinCAT UM (x86)\Release\TempContr\TempContrW32.obj"
Creating library C:\TwinCAT3\SDK_products\TwinCAT UM (x86)\Release\TempContrW32.lib and ob
TempContr.vcxproj -> C:\TwinCAT3\SDK_products\TwinCAT UM (x86)\Release\TempContrW32.dll
C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\mt.exe /nologo /verbose /out:"C:\TwinCA
Done building project "TempContr.vcxproj".
Project "TempContr.vcxproj" (TcPublishAdditionalFiles target(s)):
Done building project "TempContr.vcxproj".
Done building project "TempContr.vcxproj".
========== TwinCAT Publish Modules finished at 04.09.2012 12:11:29 ==========

Die binären Dateien und die TMC-Modulbeschreibung werden in den Ordner „TempContr“ unter „C:\TwinCAT\3.x\CustomConfig\Modules“ exportiert.

4. Kopieren Sie für den Import lediglich den Ordner „TempContr“ auf jede andere TwinCAT 3 Maschine.



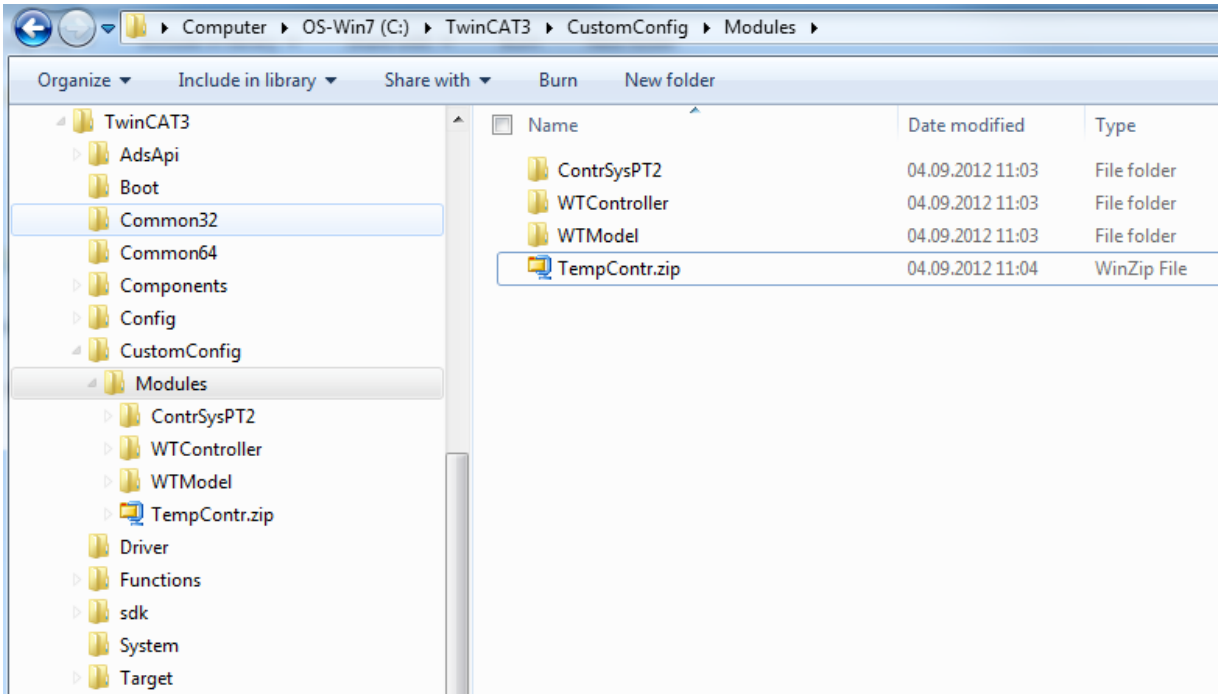
7.2 Module importieren

Szenario

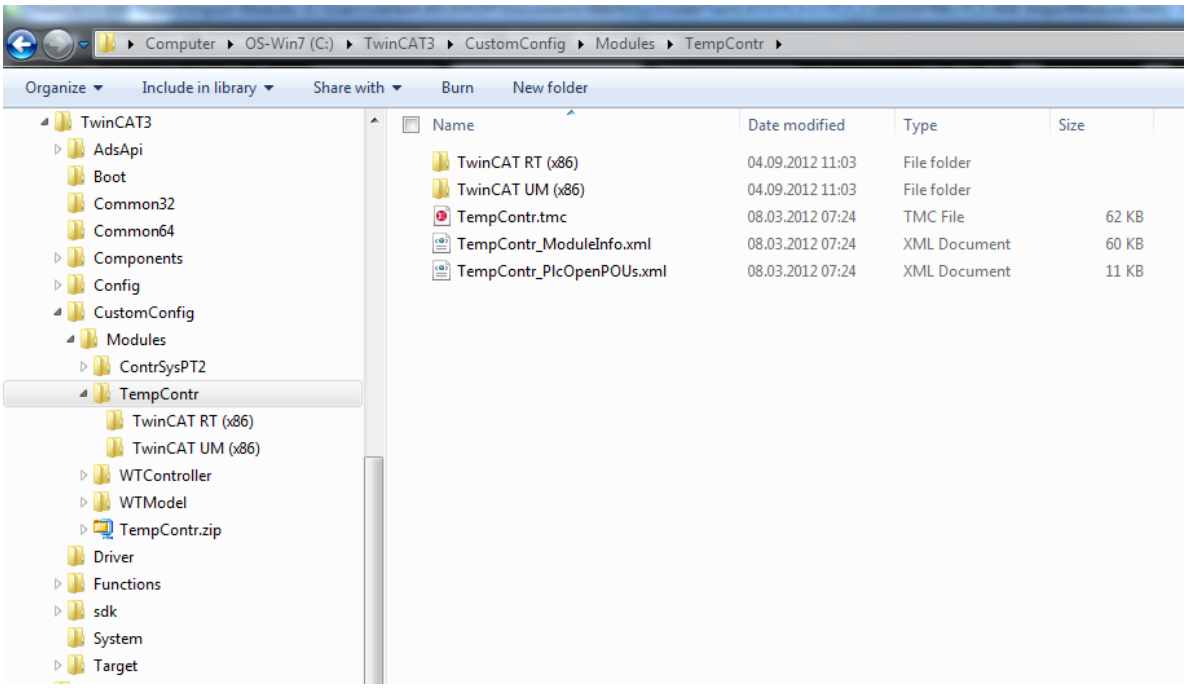
In diesem Artikel ist beschrieben, wie ein binäres TC3-Modul in eine „PC/IPC“-Steuerung mit TwinCAT 3 XAE (ohne Vollversion von Visual Studio), importiert und integriert werden kann.
Das binäre TC3-Modul wurde vorher auf einem anderen PC implementiert und exportiert.

Folgende Schritte sind zu erledigen

1. Auf dem zweiten IPC mit TwinCAT XAE ohne Vollversion von Visual Studio binäres Modul in den Zielordner „..\TwinCAT\3.x\CustomConfig\Modules“ kopieren. In diesem Beispiel wird das Archiv „TempContr.zip“ entpackt.

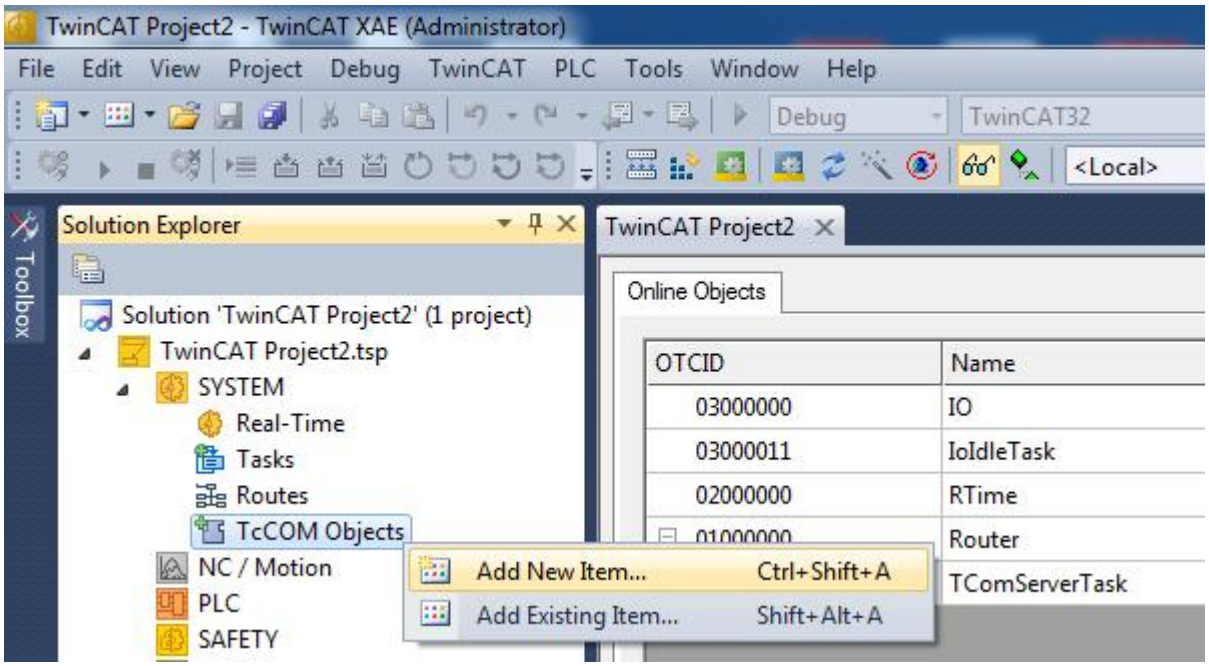


⇨ Daraufhin stellt das „TempContr“-Modul die binären Module (in den Unterordnern RT und UM) und die entsprechende TwinCAT Modul Class *.tmc Datei „TempContr.tmc“ zur Verfügung.



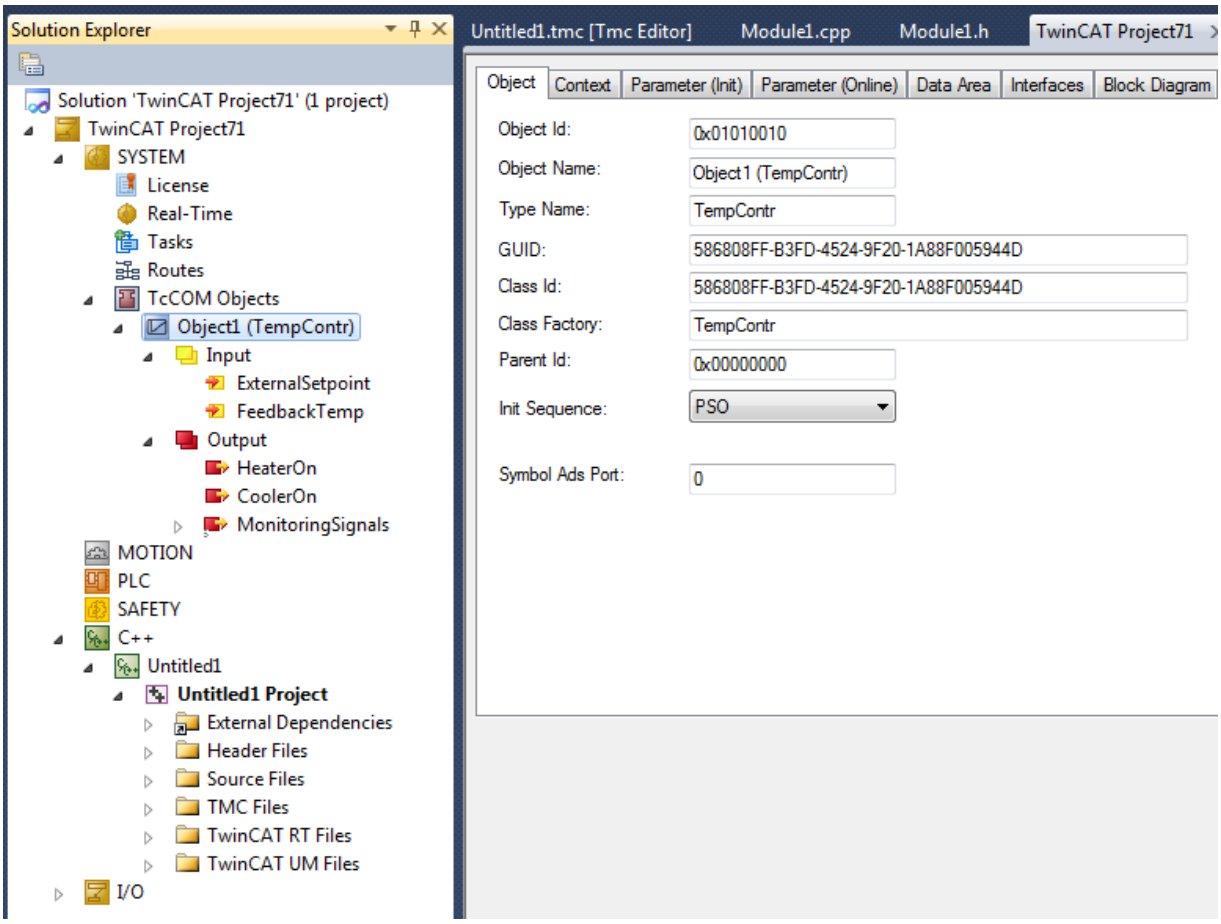
2. Starten Sie die TwinCAT XAE Umgebung und erstellen ein TwinCAT 3 Projekt.

3. Klicken Sie mit der rechten Maustaste auf **„System->TcCOM Objects“** und wählen **„Add New Item...“** aus.



4. Im eingeblendeten Dialog wird das neue TempContr Modul aufgelistet. Erstellen Sie eine Modulinstanz, indem Sie den Modulnamen auswählen und mit **„OK“** fortfahren.
5. Die Instanz des **TempContr** Moduls erscheint nun unter **„TcCom Objects“**.
   Die nächsten Schritte wie gehabt:
6. Einen neuen Task erzeugen
7. Zu „Context“ der Modulinstanz gehen und C++ Modulinstanz mit dem zuvor hinzugefügten **„Task 1“** verknüpfen.

8. Konfiguration aktivieren



8 Entwicklung

In diesem Abschnitt werden die Prozesse für Programmierung, Kompilierung und Starten eines TwinCAT C++ Projekts beschrieben.

Er soll einen allgemeinen Überblick über den Engineering-Prozess für TwinCAT C++ Projekte mit Verweisen auf die entsprechende ausführliche Dokumentation geben. Der Schnellstart [▶ 37] geht die gemeinsamen Schritte im Einzelnen durch.

1. Typen-Deklaration und Modultyp:
   Der TwinCAT Module Class Editor (TMC) [▶ 69] und TMC Code Generator werden für die Definition von Datentypen und Schnittstellen, und auch der Module, die diese verwenden, herangezogen.
   Der TMC Code Generator generiert Quellcode anhand der bearbeiteten TMC-Datei und bereitet Datentypen / Schnittstellen vor, die in anderen Projekten (wie SPS) zu verwenden sind.
   Bearbeiten und den Code Generator starten kann immer und immer wieder gemacht werden: Die Code-Generierung achtet auf programmierten Benutzercode und bewahrt diesen.
2. Programmierung
   Die bekannte Visual Studio C++ Programmierungsumgebung wird für die Entwicklung und das Debugging [▶ 60] des benutzerdefinierten Codes innerhalb der Code-Vorlage verwendet.
3. Module [▶ 23]instanziieren
   Das Programm beschreibt eine Klasse, die als Objekte instanziiert wird. Der TwinCAT Module Instance Configurator [▶ 112] wird für die Konfiguration der Instanz verwendet. Allgemeine Konfigurationselemente sind: Task zuweisen, Symbolinformation für Laufzeit (TwinCAT Module Instance (TMI) Datei) herunterladen oder Parameter- /Schnittstellenzeiger festlegen.
4. Mapping von Variablen
   Die Ein- und Ausgangsvariablen eines Objekts können mit Variablen von anderen Objekten oder SPS-Projekten unter Verwendung des normalen TwinCAT Systemmanagers verknüpft werden.
5. Erstellung (Building)
   Bei der Erstellung (Kompilieren und Verknüpfung) des TwinCAT C++ Projekts werden alle Komponenten für die ausgewählte Plattform kompiliert. Die Plattform wird automatisch festgelegt, wenn das Zielsystem ausgewählt ist.
6. Veröffentlichung (Publishing) (siehe Module exportieren [▶ 31] / Module importieren [▶ 32])
   Die Veröffentlichung eines Moduls erstellt die Treiber für alle Plattformen und bereitet es für die Verteilung vor. Das erzeugte Verzeichnis kann verteilt werden, ohne dass der Quellcode übergeben werden muss. Es wird ausschließlich Binärcode mit Schnittstellenbeschreibung übergeben.
7. Signatur (siehe x64: Treibersignierung [▶ 16])
   Die TwinCAT-Treiber müssen für x64-Laufzeiten signiert sein, weil 64-Bit-Windows Versionen fordern, dass Kernel-Module signiert sind. Also gilt dies sowohl für die x64-Erstellung als auch für das Veröffentlichen von Modulen, weil diese Module die x64-Binärcodes enthalten (falls nicht deaktiviert, wie hier [▶ 185] beschrieben).
   Der Signaturprozess kann wie hier [▶ 21] beschrieben benutzerdefiniert sein.
8. Aktivierung
   Der TwinCAT C++ Treiber kann wie jedes TwinCAT Projekt mittels „Activate Configuration“ aktiviert werden. Daraufhin bittet ein Dialog darum, TwinCAT in den RUN-Modus zu setzen.
   Das (von IEC61131-basierten Systemen her bekannte) Debuggen [▶ 60] in Echtzeit, sowie das Setzen von (bedingten) Haltepunkten ist im Falle von TwinCAT C++ Modulen möglich.

⇨ Das Modul läuft unter Echtzeitbedingungen.

9 Schnellstart

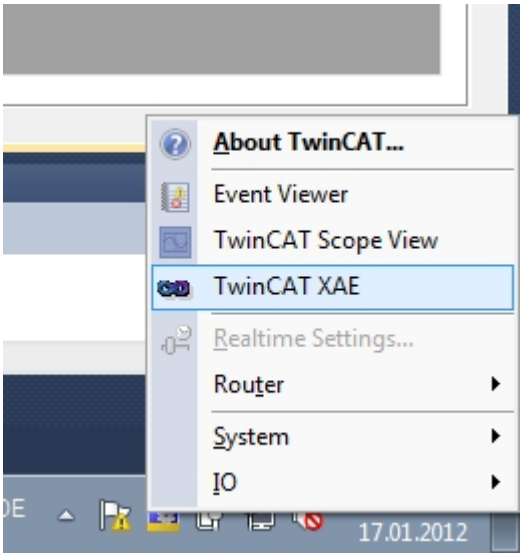
Dieser Schnellstart zeigt, wie man in kurzer Zeit mit dem TwinCAT C++ Modul Engineering vertraut werden kann.

Jeder Schritt der Erstellung eines im Echtzeitkontext laufenden Moduls ist ausführlich beschrieben.

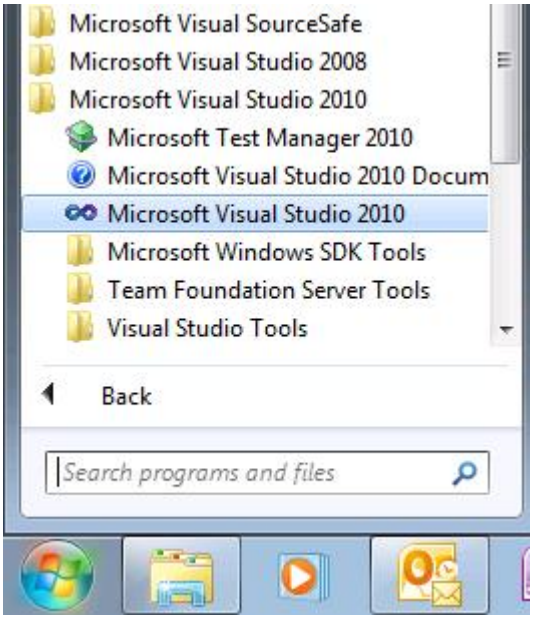
9.1 TwinCAT 3 Projekt erstellen

Engineering Umgebung starten: Visual Studio

1. „Microsoft Visual Studio“ kann auf zwei verschiedene Arten gestartet werden:
   - Option1: Rechtsklicken auf das TwinCAT 3 Symbol in der Taskleiste und „Microsoft Visual Studio“ auswählen



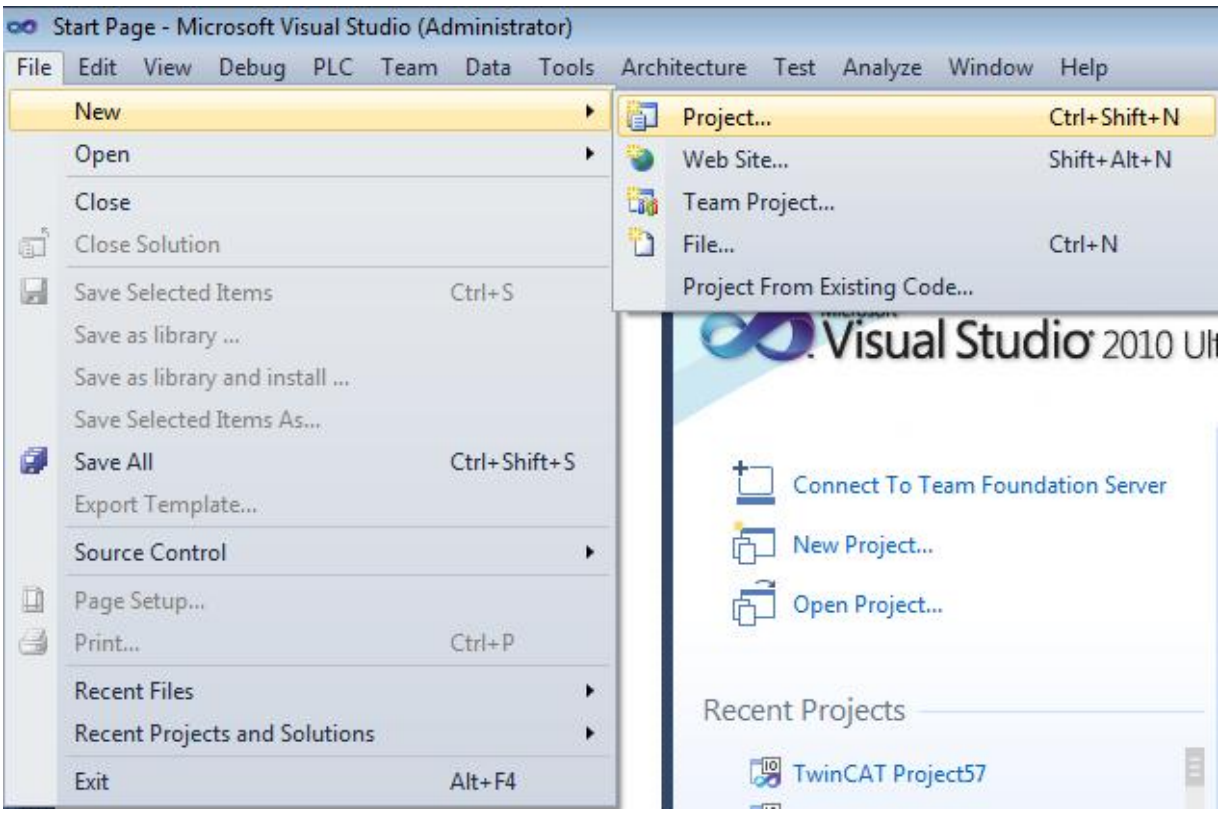
   - Option 2: Microsoft Visual Studio Umgebung über „Start->All Programs->Microsoft Visual Studio“ starten.



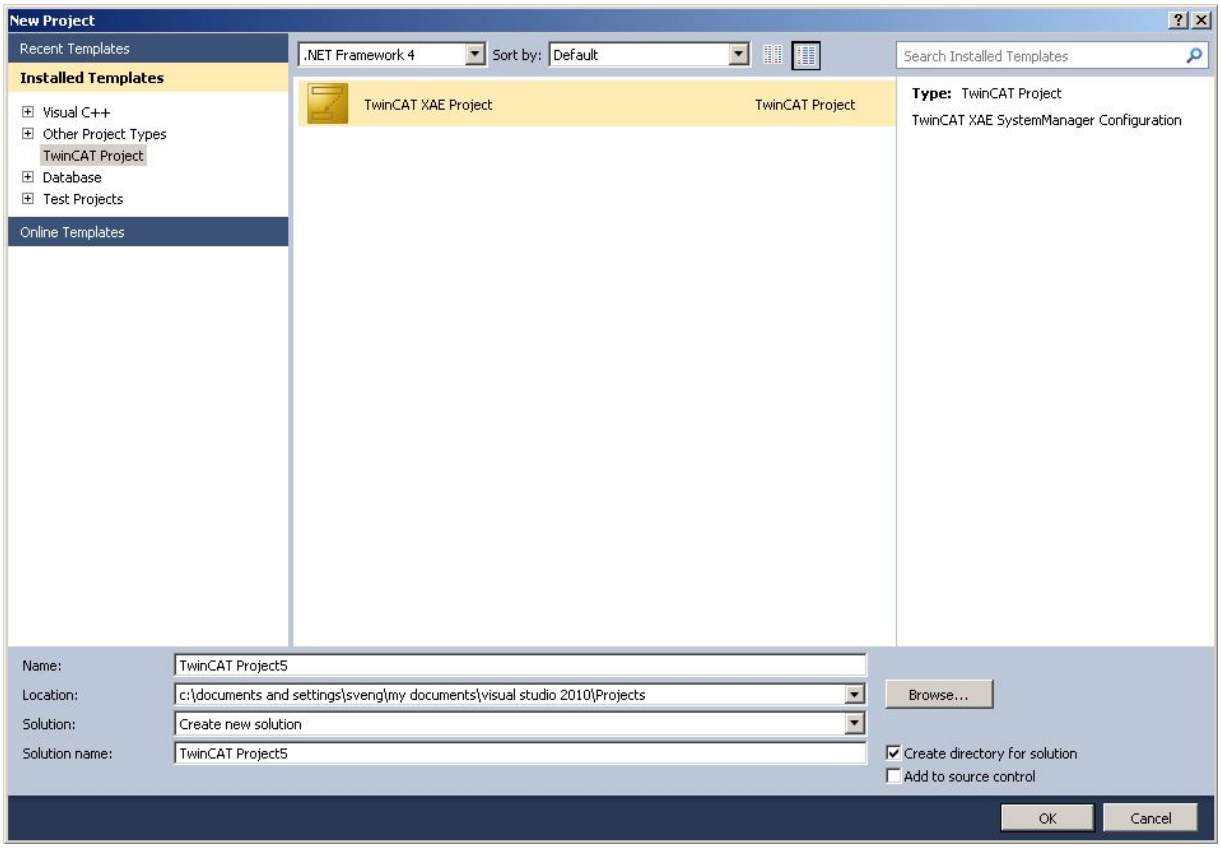
TwinCAT 3 Projekt erstellen

Um eine TwinCAT 3 Solution zu erstellen, Visual Studio öffnen und folgende Schritte ausführen:

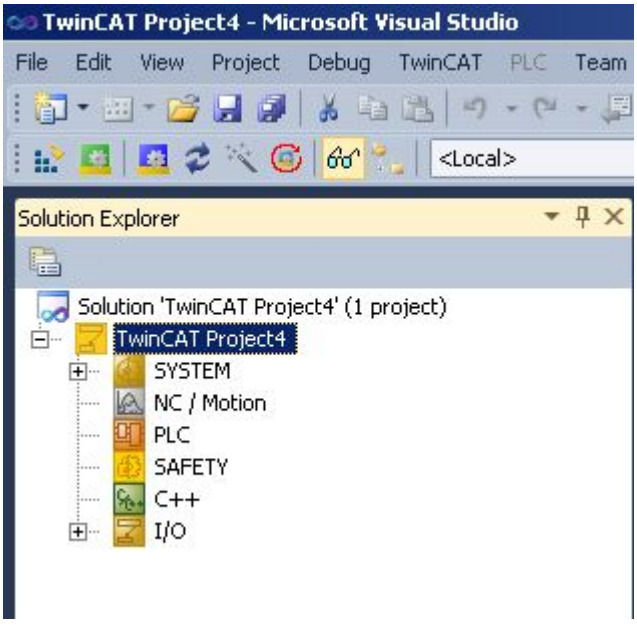
2. Auswählen: File -> New -> Project



3. Es werden alle bestehenden Projektvorlagen angezeigt. Wählen Sie „TwinCAT XAE Project“, optional kann ein passender Projektname eingegeben und mit „OK“ fortgefahren werden. Sie können den Namen des Verzeichnisses ab sofort nicht mehr wählen oder umbenennen. Behalten Sie die Standardeinstellungen bei (gewählte Option „Create directory for solution“).



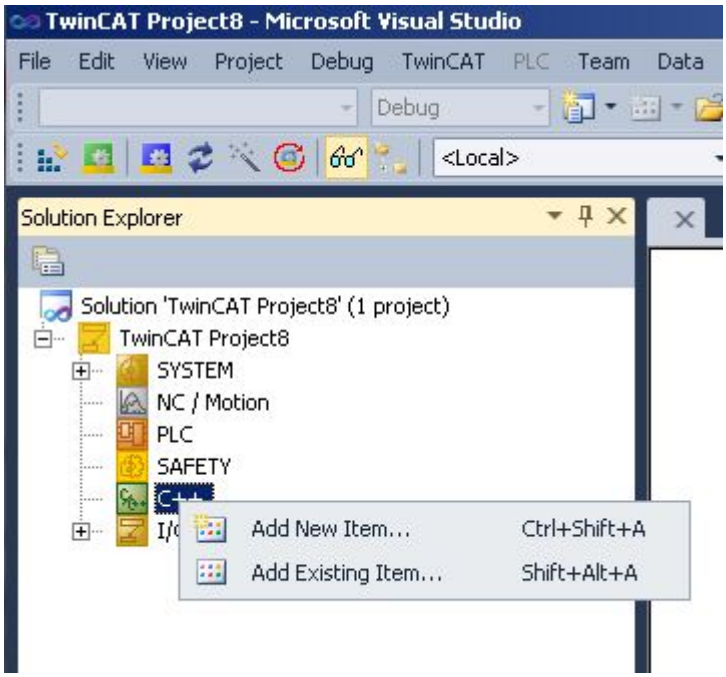
⇨ Daraufhin zeigt der Visual Studio Solution Explorer das TwinCAT 3 Projekt.



9.2 TwinCAT 3 C++ Projekt erstellen

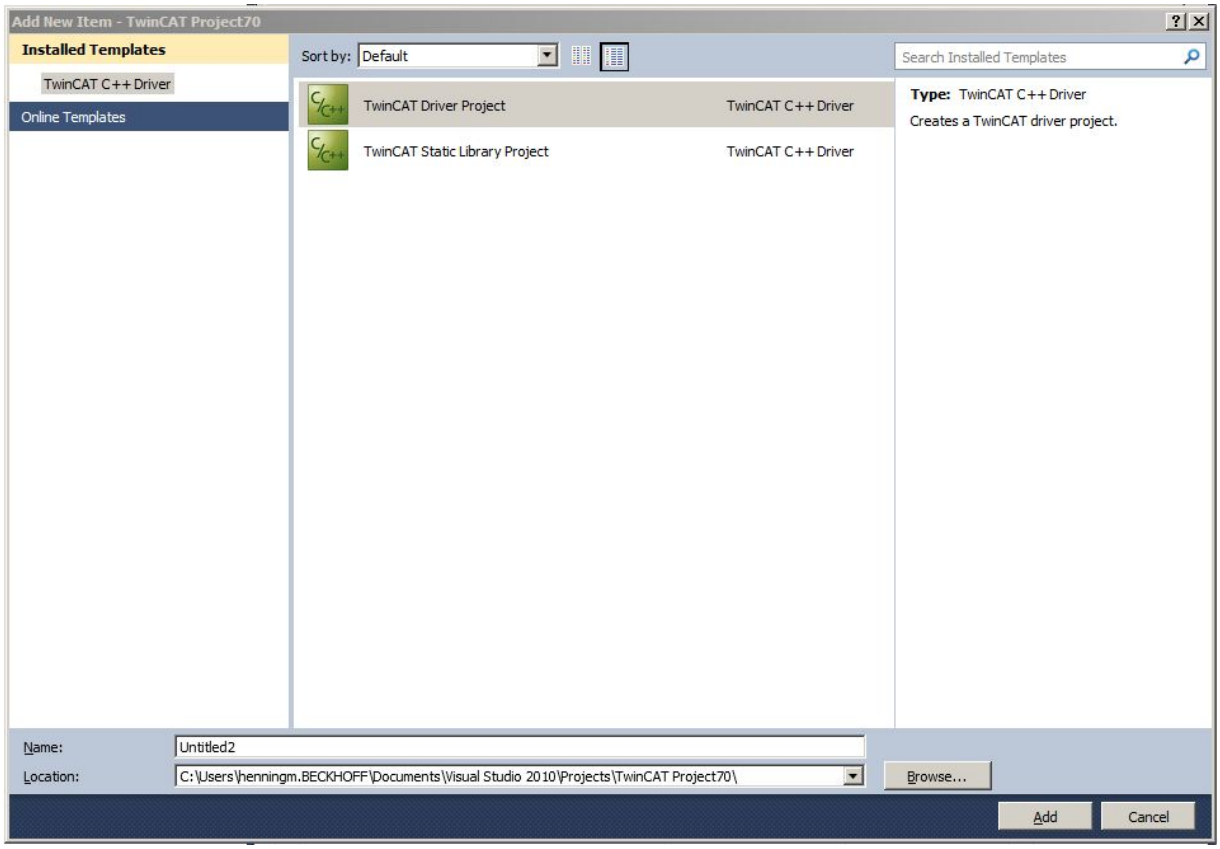
Nach der Erstellung eines TwinCAT 3 Projekts, den „C++“ Knoten öffnen und den folgenden Schritten folgen:

1. Klicken Sie mit der rechten Maustaste auf „C++“ und „Add New Item...“ auswählen.
   Wenn das grüne C++ Symbol nicht aufgeführt ist, dann wird die Installation von Microsoft Visual Studio 2010, Level Professional, Premium oder Ultimate nicht erkannt und Visual Studio Shell läuft.
   Bitte Visual Studio Professional / Premium oder Ultimate (erneut) installieren.



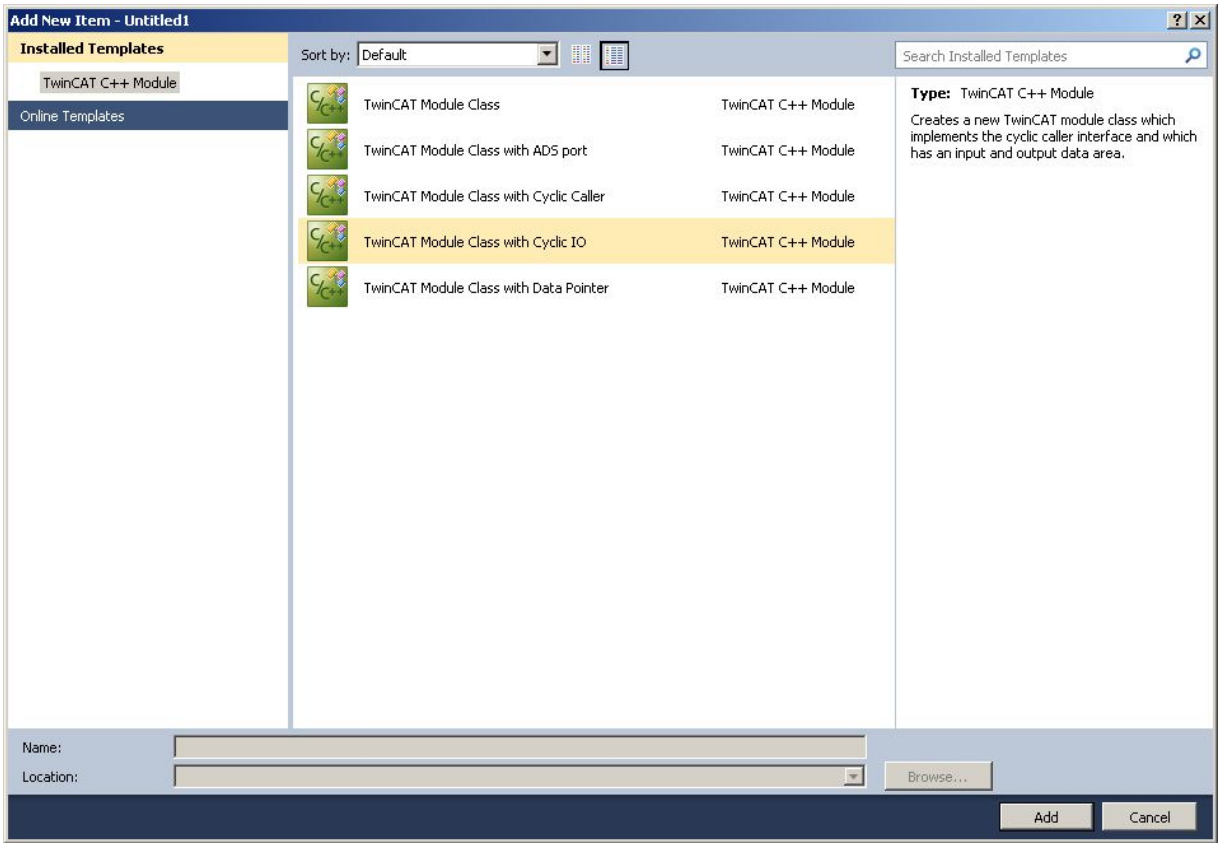
⇨ Der „TwinCAT C++ Projekt-Assistent [▶ 65]“ wird eingeblendet.

2. Es werden alle bestehenden Projektvorlagen aufgeführt, normalerweise ist „TwinCAT Driver Project“ auszuwählen, optional kann ein verwandter Projektnamen eingegeben und mit „OK“ fortgefahren werden. Alternativ kann das „TwinCAT Static Library Project“ verwendet werden, das eine Umgebung für das Programmieren von statischen TC-C++ Bibliotheken bereitstellt (siehe Beispiel25 [▶ 252])



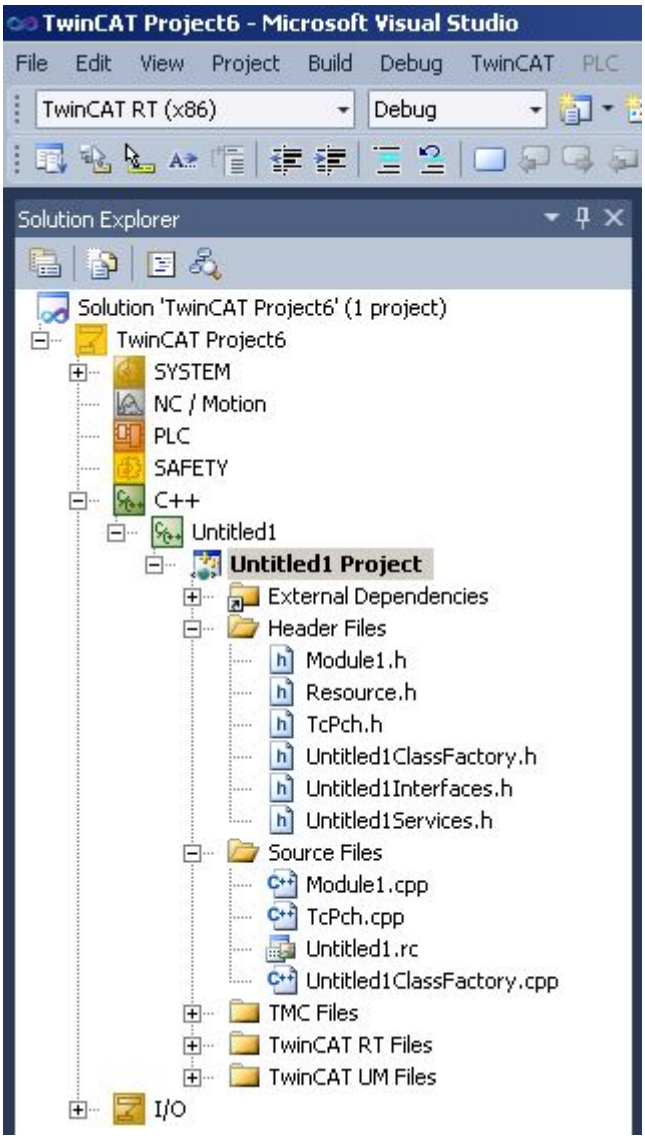
⇨ Der „TwinCAT Modul-Assistent [▶ 66]“ wird eingeblendet.

3. Wählen Sie aus den folgenden Vorlagen aus:
   „TwinCAT Module Class“ - erzeugt eine neue TwinCAT-Modulklasse.
   „TwinCAT Module Class with Cyclic Caller“ - erzeugt eine neue TwinCAT-Modulklasse, die eine zyklisch aufrufende Schnittstelle implementiert.
   „TwinCAT Module Class with Cyclic IO“ - erzeugt eine neue TwinCAT-Modulklasse, die die zyklisch aufrufende Schnittstelle implementiert und die einen Ein- und Ausgangsdatenbereich aufweist.
   „TwinCAT Module Configuration Wizard“ - ermöglicht die Bearbeitung von bestehenden Modulen.
4. Wählen Sie in diesem Falle „TwinCAT Module Class with Cyclic I/O“ aus und fahren Sie mit „OK“ fort. Ein Name ist nicht erforderlich und kann hier auch nicht eingegeben werden.


Add New Item - Untitled1
Installed Templates
TwinCAT C++ Module
Online Templates
Sort by: Default
Search Installed Templates
TwinCAT Module Class
TwinCAT Module Class with ADS port
TwinCAT Module Class with Cyclic Caller
TwinCAT Module Class with Cyclic IO
TwinCAT Module Class with Data Pointer
TwinCAT C++ Module
Type: TwinCAT C++ Module
Creates a new TwinCAT module class which implements the cyclic caller interface and which has an input and output data area.
Name:
Location:
Browse...
Add
Cancel

5. Geben Sie im Dialogfenster „TwinCAT Class Wizard“ einen eindeutigen Namen ein oder fahren Sie mit dem Vorschlag „Object1“ fort.

⇨ Daraufhin wird ein eines TwinCAT 3 C++ Projekt erstellt

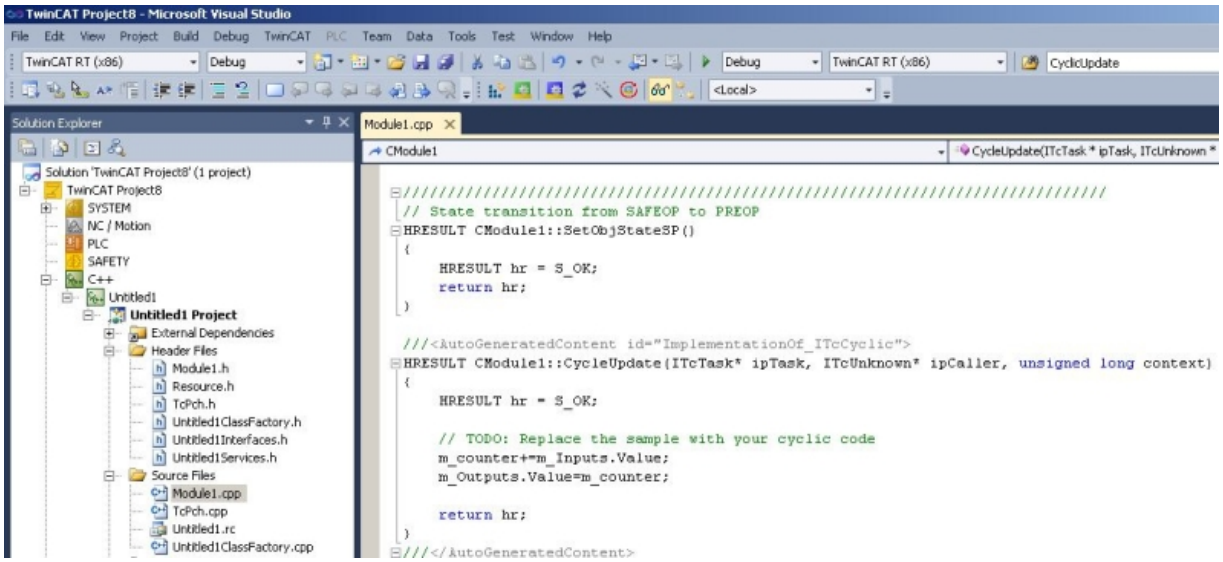


9.3 TwinCAT 3 C++ Projekt implementieren

In diesem Artikel wird beschrieben, wie das Beispielprojekt geändert werden kann.

Nach der Erstellung eines TwinCAT C++ Projekts und dem Öffnen der von „<MyClass>.cpp“ (in diesem Beispiel „Module1.cpp“) beginnt die Implementierung.

1. Die Methode <MyClass>::CycleUpdate() wird zyklisch aufgerufen - das ist die Stelle, wo die zyklische Logik zu positionieren ist. An dieser Stelle muss der gesamte zyklische Code hinzugefügt werden.



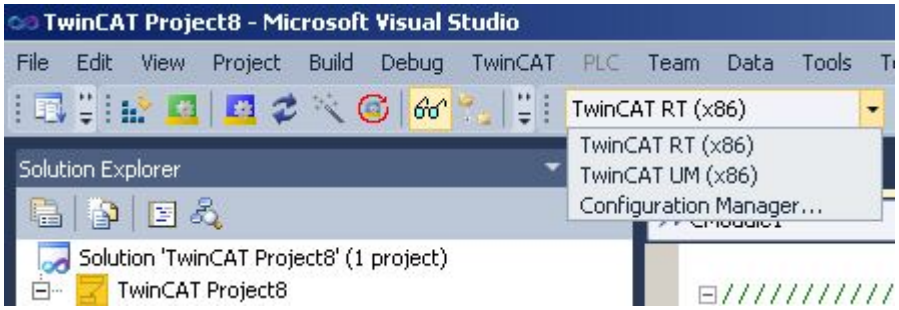
2. In diesem Falle wird ein Zähler um den Wert der Variablen „Value“ im Logik-Eingangsabbild inkrementiert.
   Ersetzen Sie eine Zeile, um den Zähler zu inkrementieren, ohne Abhängigkeit des Werts vom Logik-Eingangsabbild.
   Diese Zeile
   `m_counter+=m_Inputs.Value;`
   durch diese ersetzen
   `m_counter++;`
3. Speichern Sie die Änderungen

9.4 Compile/Build a TwinCAT 3 C++ project

In diesem Artikel wird beschrieben, wie eine bereits implementierte C++ Modulklasse erstellt wird.

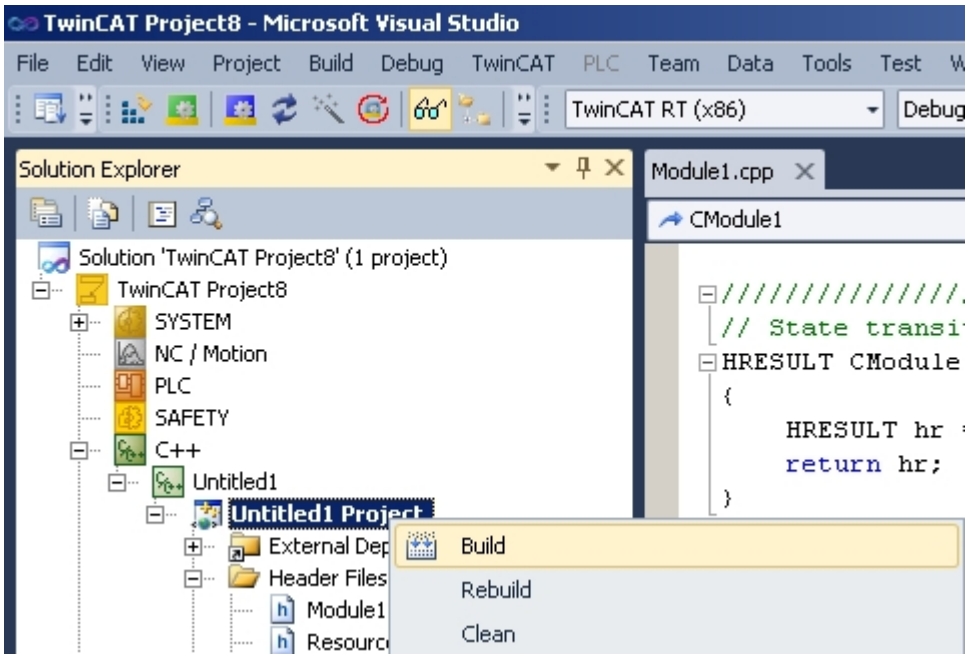
Nach der Implementierung der zyklischen benutzerdefinierten Logik in das TwinCAT C++ Projekt muss das Projekt nach jeder Code-Änderung kompiliert werden.

1. Wählen Sie die Zielplattform aus, entsprechend der die Kompilierung vorgenommen werden soll. „TwinCAT RT (x86)“ ist die standardmäßige - und sie kann auch für dieses Beispiel beibehalten werden.

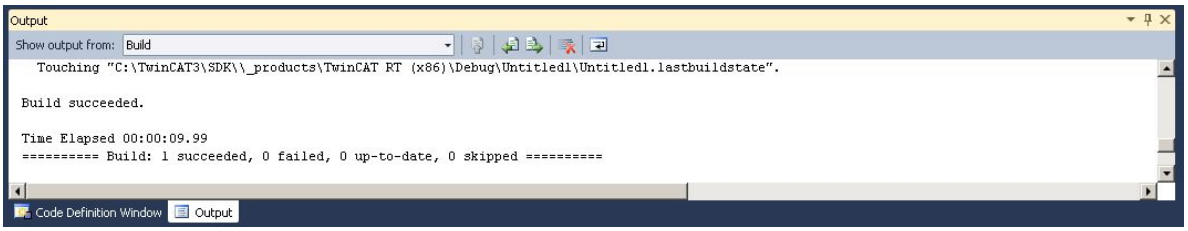

TwinCAT Project8 - Microsoft Visual Studio
File Edit View Project Build Debug TwinCAT PLC Team Data Tools
TwinCAT RT (x86)
TwinCAT RT (x86)
TwinCAT UM (x86)
Configuration Manager...
Solution Explorer
Solution 'TwinCAT Project8' (1 project)
TwinCAT Project8

2. Rechtsklick auf das TwinCAT 3 C++ Projekt und Auswahl von „Build“ oder „Rebuild“

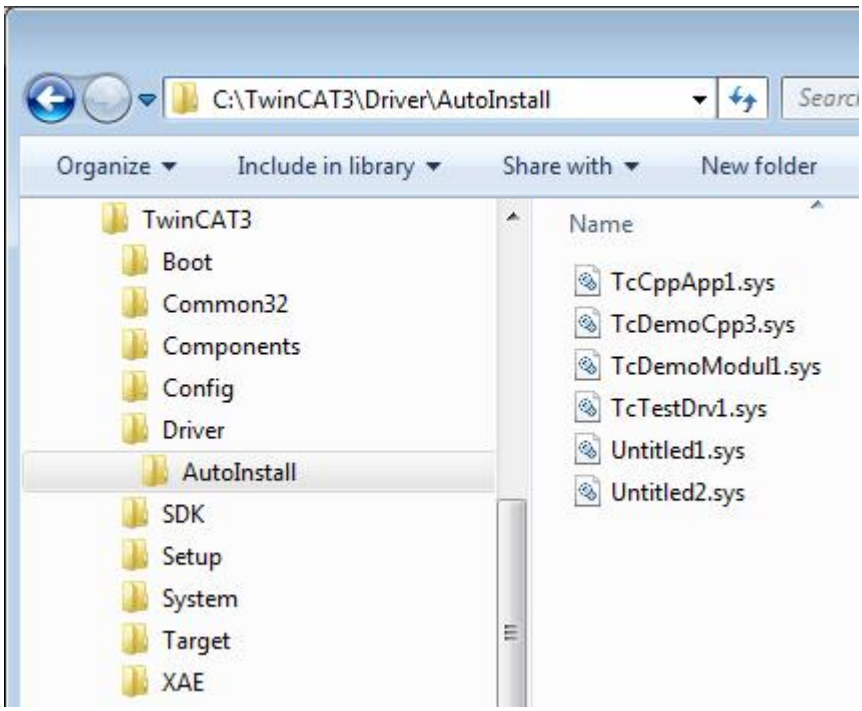
HINWEIS! In den ersten Versionen muss TwinCAT 3 im Zustand „Config-Mode“ sein.



⇨ Wenn der Code korrekt erstellt wurde (d.h. keine Syntaxfehler), dann muss das Compiler-Ausgabefenster so aussehen wie unten:



⇨ Nach erfolgreicher Kompilation/Erstellung wird das neue TwinCAT C++ Modul automatisch in das Verzeichnis „$(TwinCAT 3Dir)\Driver\AutoInstall“ bereitgestellt.

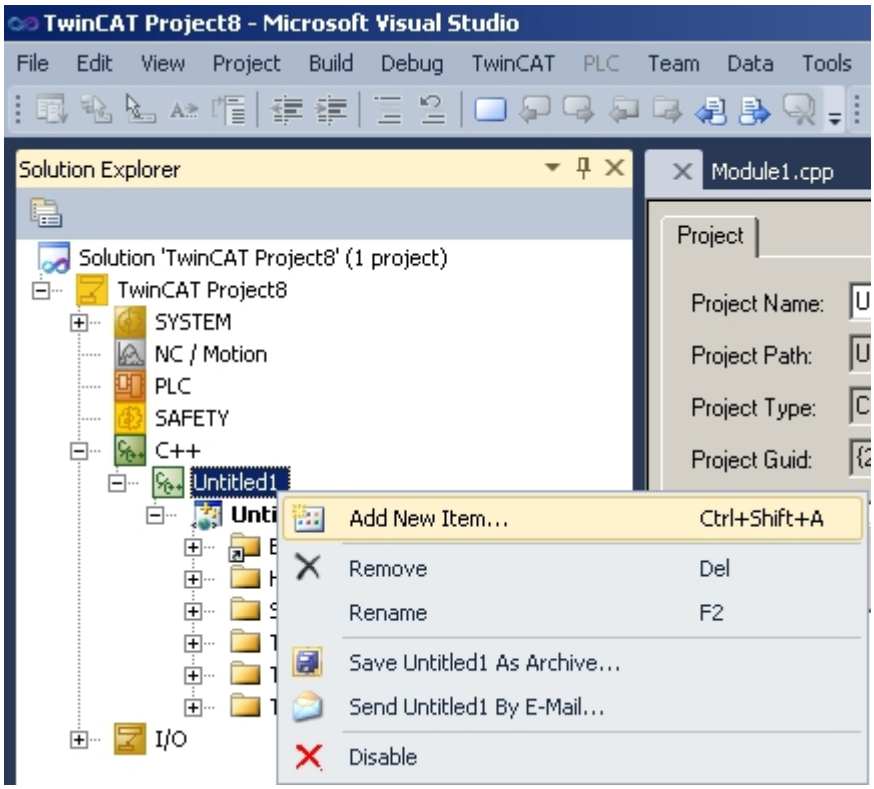


9.5 TwinCAT 3 C++ Modulinstanz erstellen

Um es ausführen zu können, muss eine Instanz des Moduls erstellt werden. Es können mehrere Instanzen eines Moduls existieren.

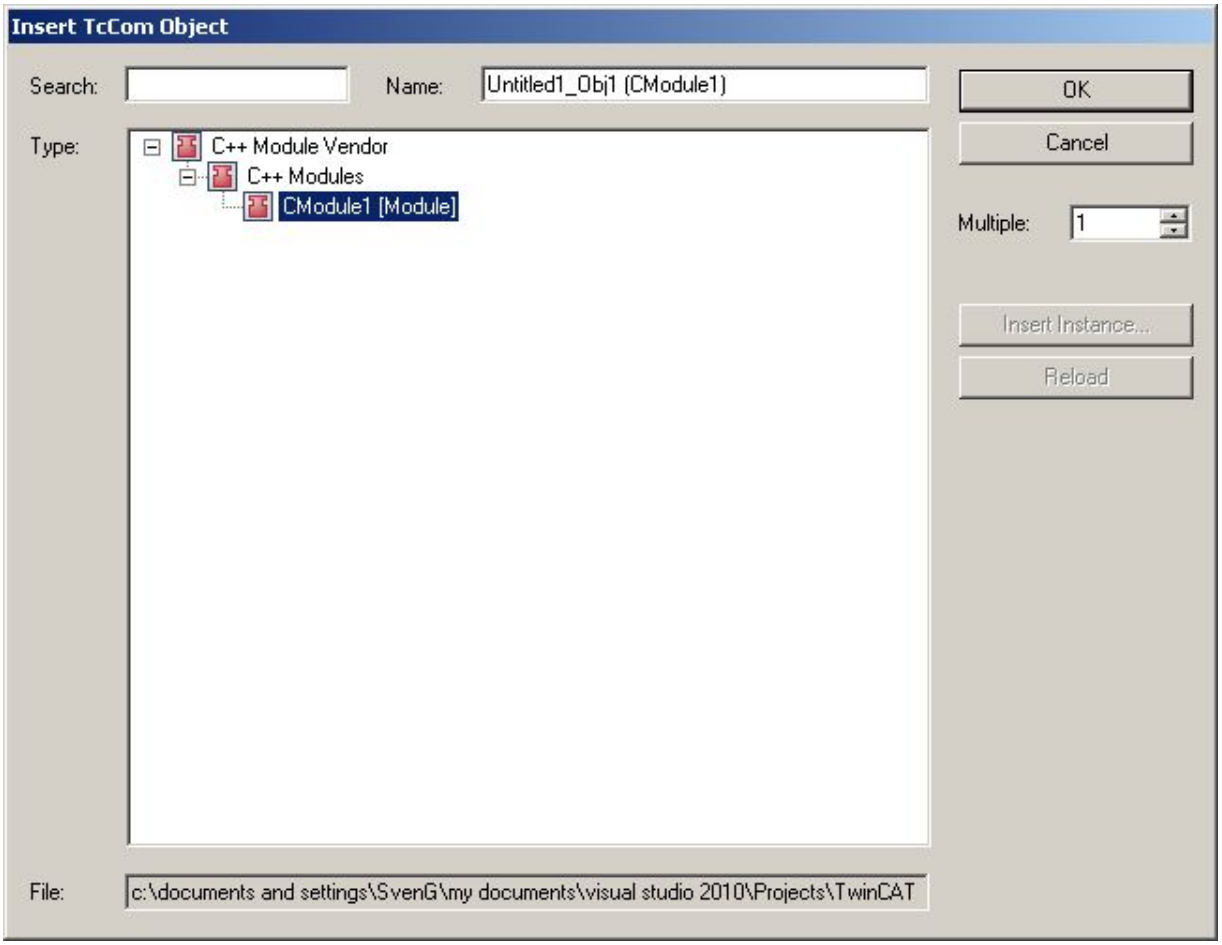
Nach Erstellen eines TwinCAT C++ Moduls, Knoten „C++ - Configuration“ öffnen und dann diesen Schritten folgen, um eine Instanz zu erstellen.

1. Rechtsklick auf das C++ Modul (in diesem Fall „Untitled1“) und Auswahl von „Add New Item...“



⇨ Alle bestehenden C++ Module werden aufgelistet.

2. Wählen Sie ein C++ Modul aus. Es kann die Standardbezeichnung verwendet oder optional ein neuer Instanzenname eingegeben und mit „OK“ bestätigt werden (in diesem Beispiel wurde die Standardbezeichnung gewählt).



⇨ Die neue Instanz „Untitled1_Obj1 (CModule1)“ wird Teil der TwinCAT 3 Solution: Der neue Knoten findet sich genau unter der TwinCAT 3-C++ Quelle „Untitled1 Project“.

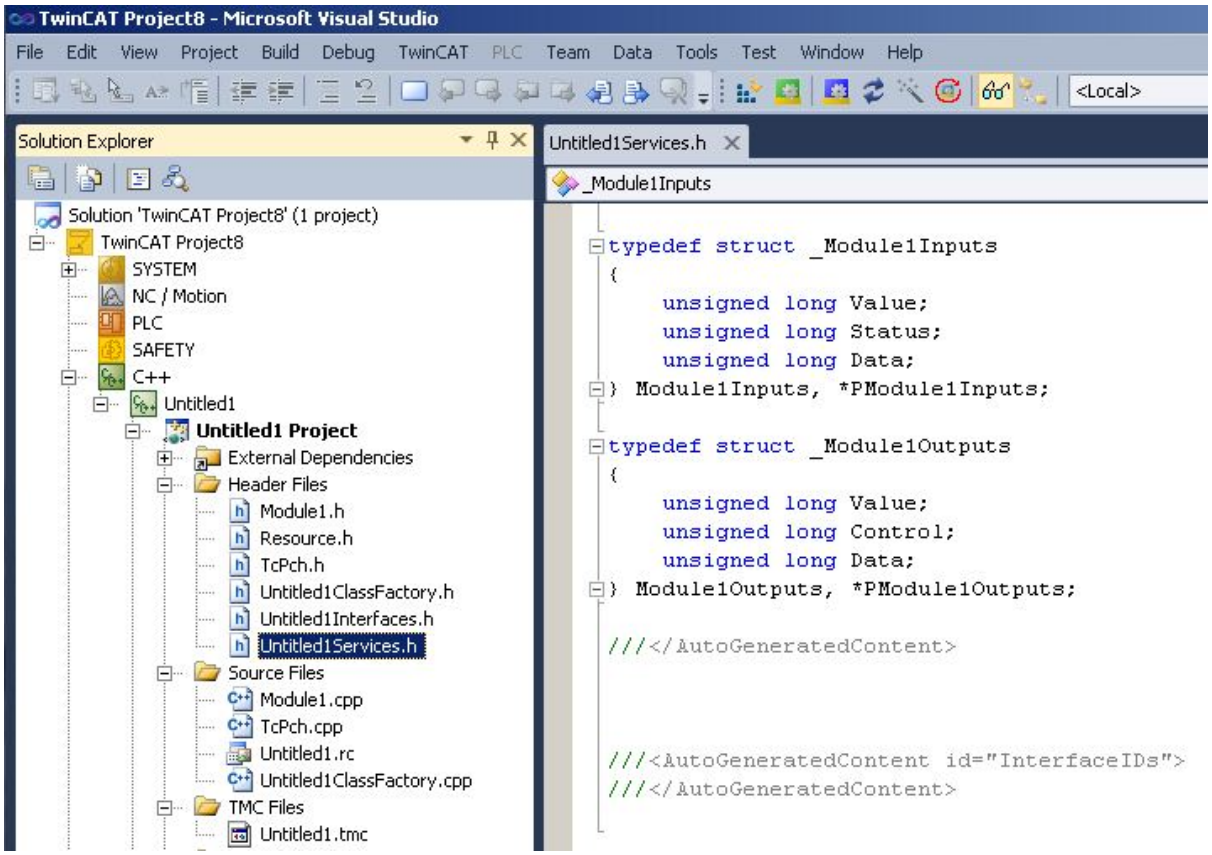
Das Modul stellt bereits eine einfache I/O-Schnittstelle mit je 3 Variablen zur Verfügung:

- Input-area: Value, Status, Data
- Output-area: Value, Control, Data

Die Beschreibung dieser Schnittstellen entspricht einander an zwei Stellen:

- „<Classname>Services.h“ (in diesem Beispiel „Untitled1Services.h“)

- „TwinCAT Module Configuration“ tmc-Datei (in diesem Beispiel „Untitled1.tmc“)



9.6 Einen Task erstellen, um eine TwinCAT 3 C++ Modulinstanz aufzurufen

Bevor ein Aufruf für eine TwinCAT 3 C++ Modulinstanz konfiguriert werden kann, muss es eine Modulinstanz geben. Um eine Modulinstanz zu erzeugen folgen Sie den ersten Schritten zur Erzeugung einer TwinCAT 3 C++ Modulinstanz.

Übersicht

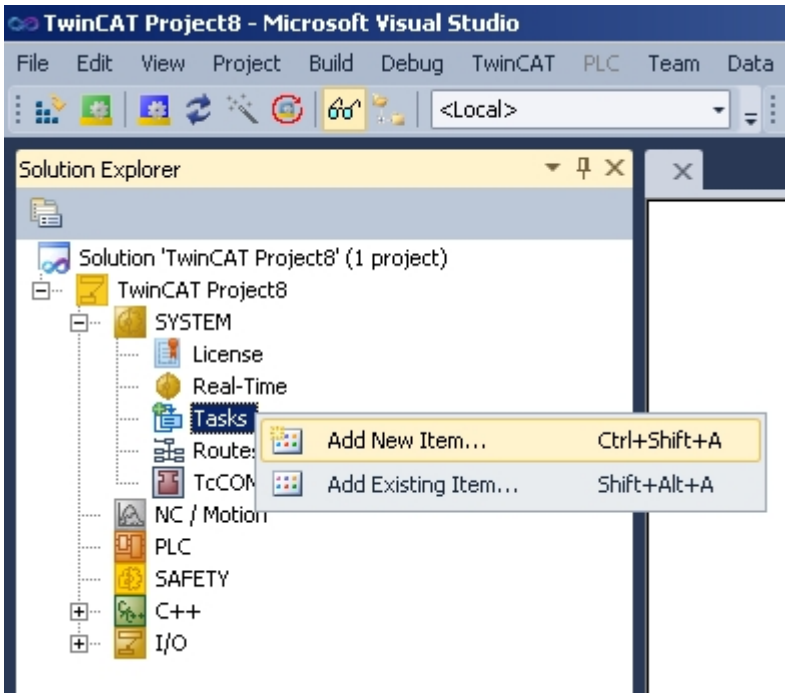
Dieser Konfigurationsschritt muss nur einmal ausgeführt werden. Für spätere Erstellungen/ Neukompilierungen des C++ Moduls muss kein neuer Task konfiguriert werden.

- Einen TwinCAT 3 „Task“ erstellen [▶ 48]: Grundlegende Aufgabe eines „Tasks“ ist es, auszuführende TwinCAT 3 Module zyklisch aufzurufen.
- TwinCAT 3 C++ Modulinstanz konfigurieren, die den TwinCAT 3 Task aufrufen soll [▶ 50]

TwinCAT 3 Task erstellen

Nach Erstellen eines TwinCAT 3 Projekts den „System configuration“-Knoten öffnen und folgende Schritte ausführen: (wenn bereits ein Task existiert, kann dieser auf Wunsch verwendet werden).

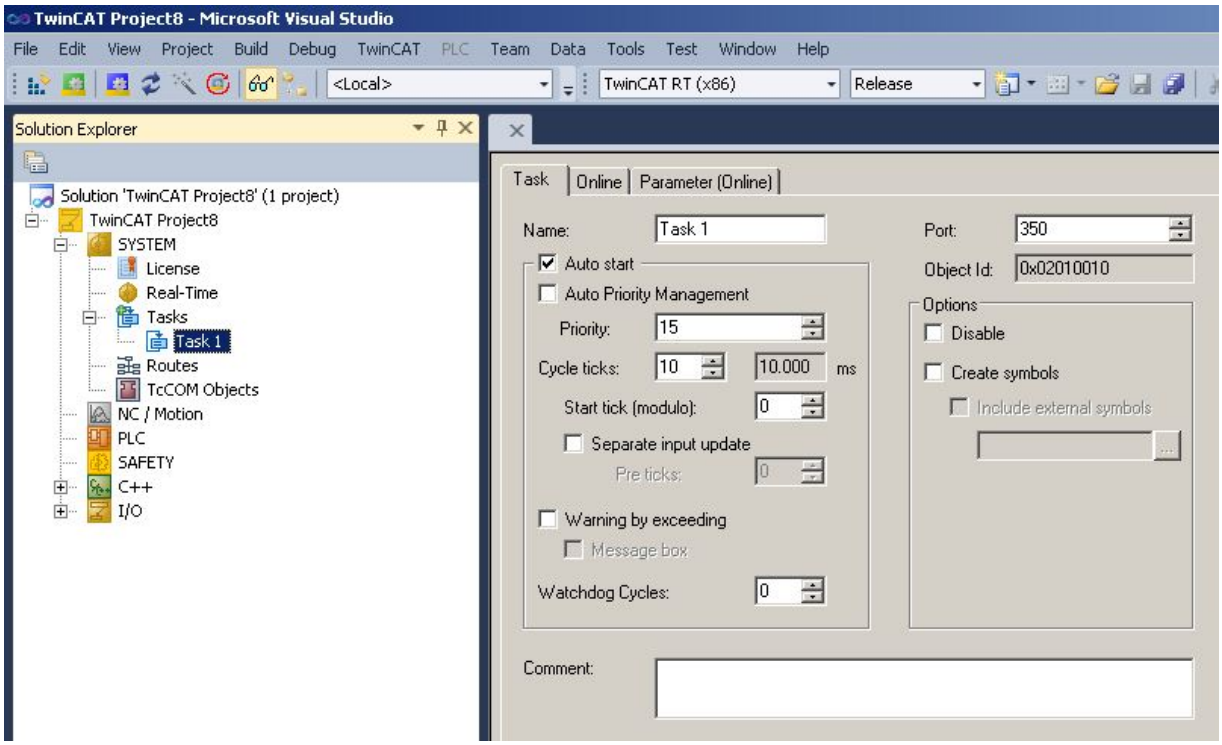
1. „System“ öffnen, Rechtsklick auf „Tasks“ und „Add New Item...“ auswählen.



2. Eindeutigen Namen eingeben (oder Standard beibehalten) und mit „OK“ fortfahren.
   HINWEIS! Standardmäßig wird kein Abbild erzeugt - optional kann das Prüfkästchen „With image“ aktiviert werden, um ein logisches Ein-/Ausgabeprozessabbild zu erhalten.
   In diesem Beispiel wird die I/O-Abbildschnittstelle durch eine C++ Modulinstanz bereitgestellt, so dass für das Auslösen der Ausführung der C++ Modulinstanz kein Abbild nötig ist.

   ⇨ Der neue Task mit Namen „Task 1“ öffnet einen Konfigurationsdialog.
3. Die wichtigsten Parameter sind „Auto start“ und „Priority“: „Auto start“ muss aktiviert werden, um einen zyklisch auszuführenden Task automatisch zu starten.
4. wählen Sie eine Priorität (in diesem Fall 15). HINWEIS: Wenn mehrere Tasks verwendet werden, sollte die höchste Priorität (kleinste Zahl) mit der kleinsten Zykluszeit verknüpft werden.

5. Legen Sie die „Cycle ticks“ fest (in diesem Fall 10ms - relativ langsam, aber für dieses Beispiel mehr als ausreichend!)

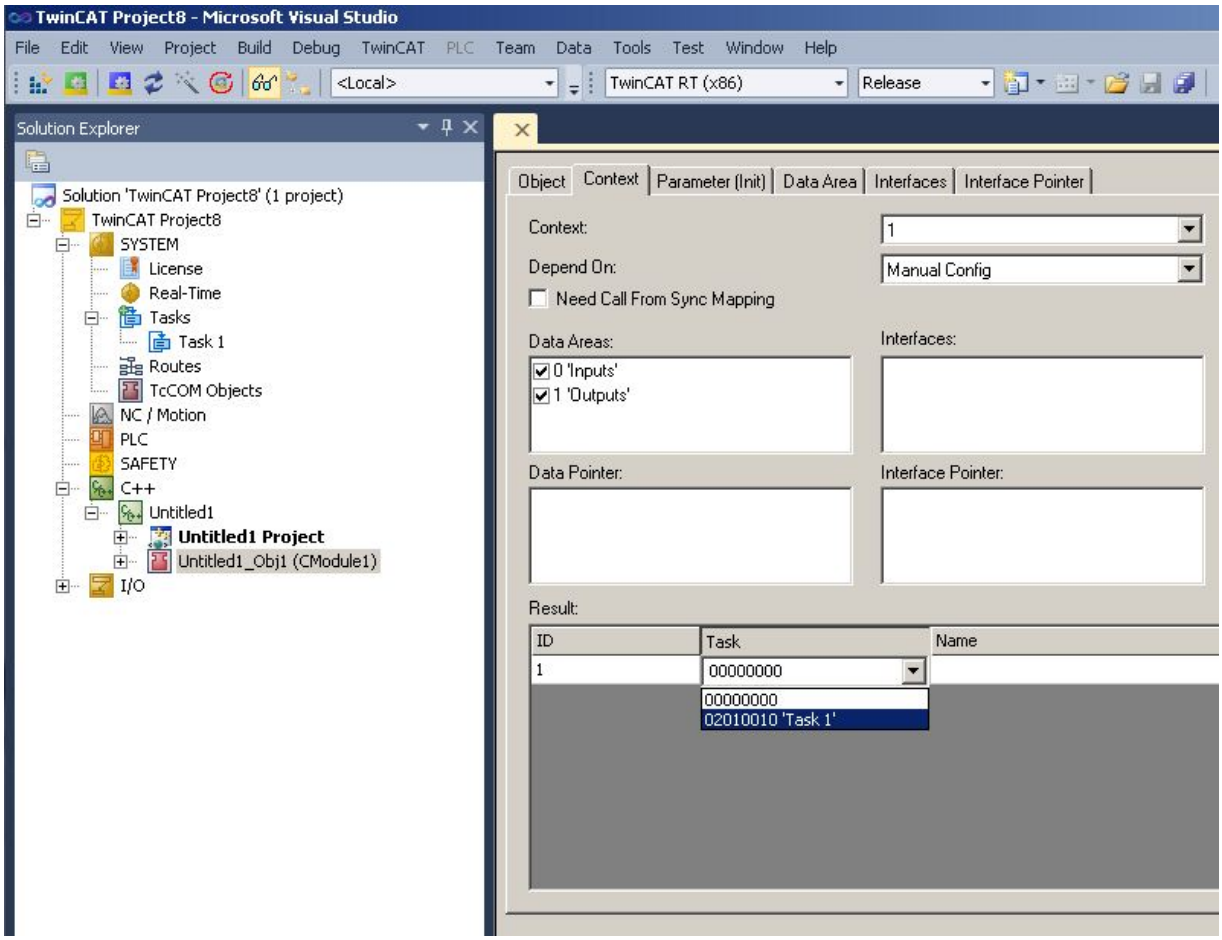


TwinCAT 3 C++ Modulinstanz konfigurieren, die von dem Task aufzurufen ist

Bis jetzt haben wir nur ein C++ Modul implementiert und eine Instanz dieses Moduls konfiguriert. Jetzt konfigurieren wir diese Instanz so, dass sie zyklisch ausgeführt wird:

6. C++ Modulinstanz im Solution-Baum wählen.
7. Im rechten Arbeitsbereich Registerkarte „Context“ wählen.

8. Task für den zuvor erstellten Kontext im Dropdown-Taskmenü wählen.
   Wählen Sie im Beispiel die Standard „Task 1“.



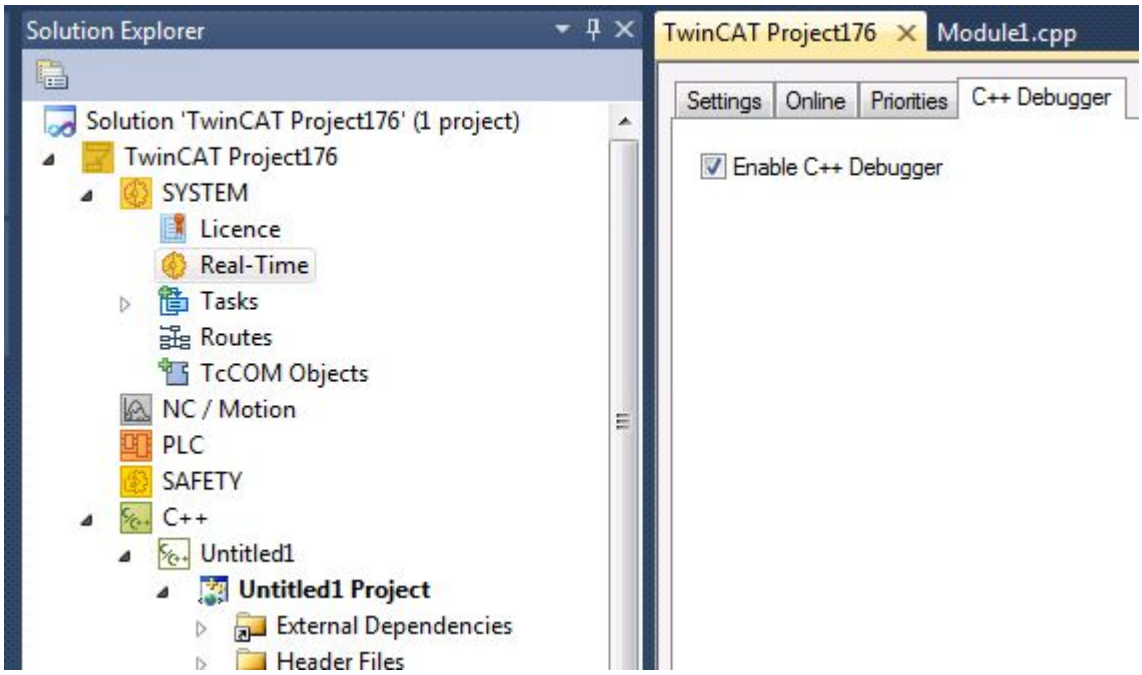
⇨ Mit Abschluss dieses Schritts ist der „Interface Pointer“ als „Cyclic Caller“ konfiguriert.

⇨ Die Konfiguration ist jetzt abgeschlossen!

9.7 TwinCAT 3 C++, C++ Debugger aktivieren

Für den C++ Debugger gibt es weitere Voraussetzungen. Um nicht immer alle diese Abhängigkeiten zu laden, muss diese Funktion einmal vor der Aktivierung der Konfiguration aktiviert werden.

1. Zu „System -> Real-Time“ navigieren, Registerkarte „C++ Debugger“ wählen und „Enable C++ Debugger“ aktivieren.

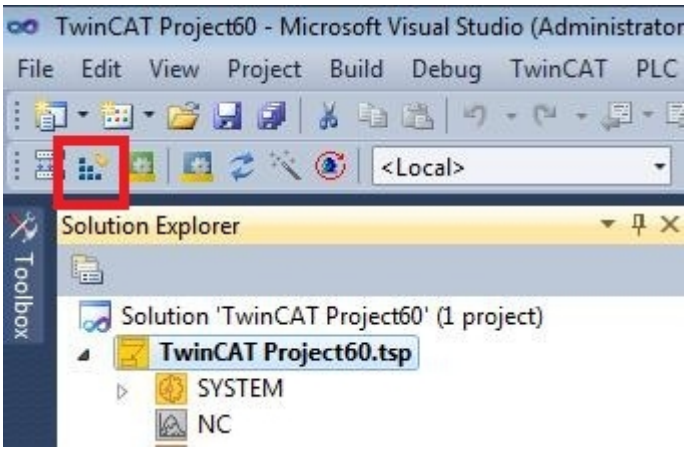


9.8 TwinCAT 3 C++ Projekt aktivieren

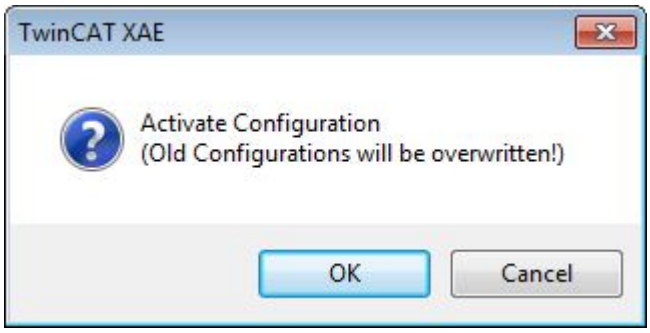
In diesem Artikel wird erläutert, wie das TwinCAT 3 C++ Beispielprojekt aktiviert und ausgeführt wird.

Nachdem ein TwinCAT C++ Projekt erstellt, kompiliert und bereitgestellt wurde, muss die Konfiguration aktiviert werden:

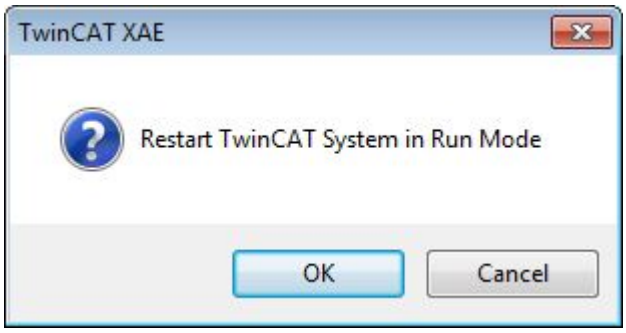
1. Klicken Sie auf das Symbol „Activate Configuration“ - die TwinCAT 3 Systemmanager Datei wird als aktuelle Konfiguration aktiviert.



2. Bestätigen Sie im nächsten Schritt die Aktivierung der neuen Konfiguration. Die vorherige alte Konfiguration wird überschrieben.



3. TwinCAT 3 fragt automatisch ob in den Run-Modus gewechselt werden soll.



⇨ Bei **„OK“** wechselt das TwinCAT 3 Projekt in den Run-Modus.
Bei **„Cancel“** bleibt TwinCAT 3 im **Config-Modus**.

⇨ Nach dem Wechsel in den Run-Modus, leuchtet das TwinCAT System Service Symbol in der Taskleiste grün auf.

9.9 TwinCAT 3 C++ Projekt debuggen

Dieser Artikel beschreibt das Debuggen des TwinCAT 3 C++ Beispielprojekts. Hauptbestandteile sind:

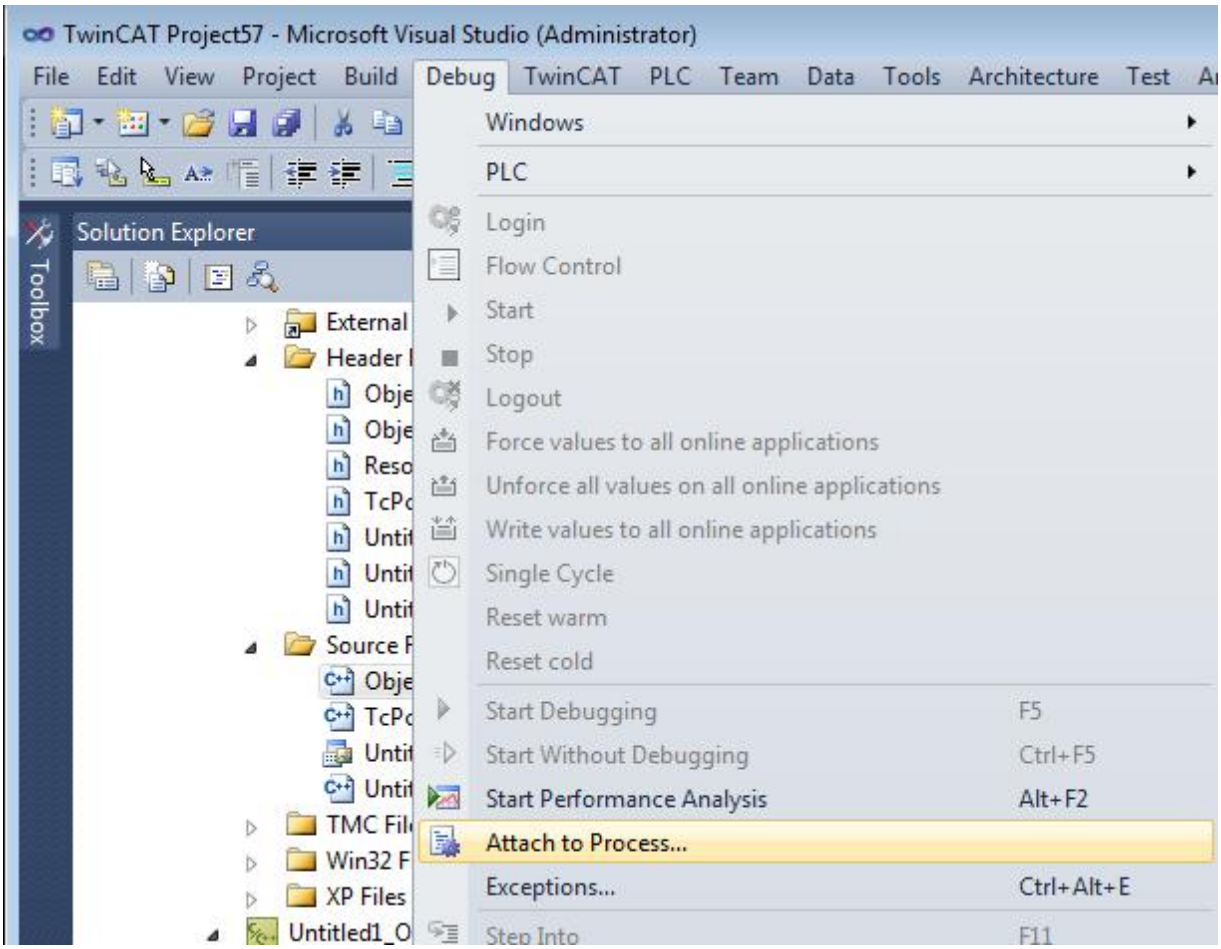
1. An C++ Modulinstanz anhängen [▶ 53]
2. Membervariablen überwachen (ohne Haltepunkte) [▶ 56]
3. Haltepunkte setzen [▶ 58]
4. Clean up: Trennen [▶ 58]

An C++ Modulinstanz anhängen

Das C++ Zielmodul kann auf zwei Arten angebunden werden:
Option1:

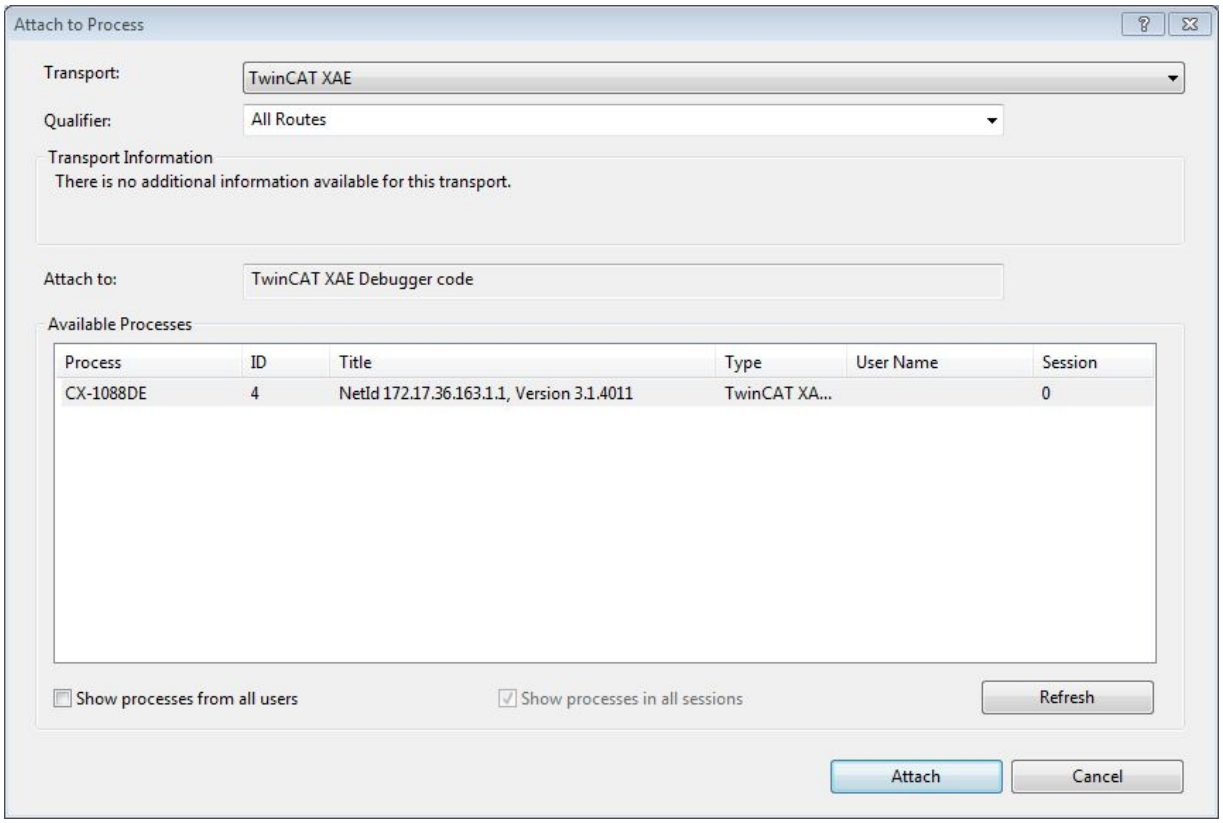
Um an das TwinCAT Modul anzuhängen

1. „Debug“ -> „Attach to process...“ in der Visual Studio Umgebung wählen:



2. Für den Transport nicht die „Default“-Einstellung von Visual Studio wählen sondern „TwinCAT XAE“
3. Zielmaschine (oder „All Routes“) als Qualifizierer wählen

4. ENTER drücken, um die Prozessliste zu aktualisieren



⇨ Die verfügbaren Prozesse werden aufgelistet

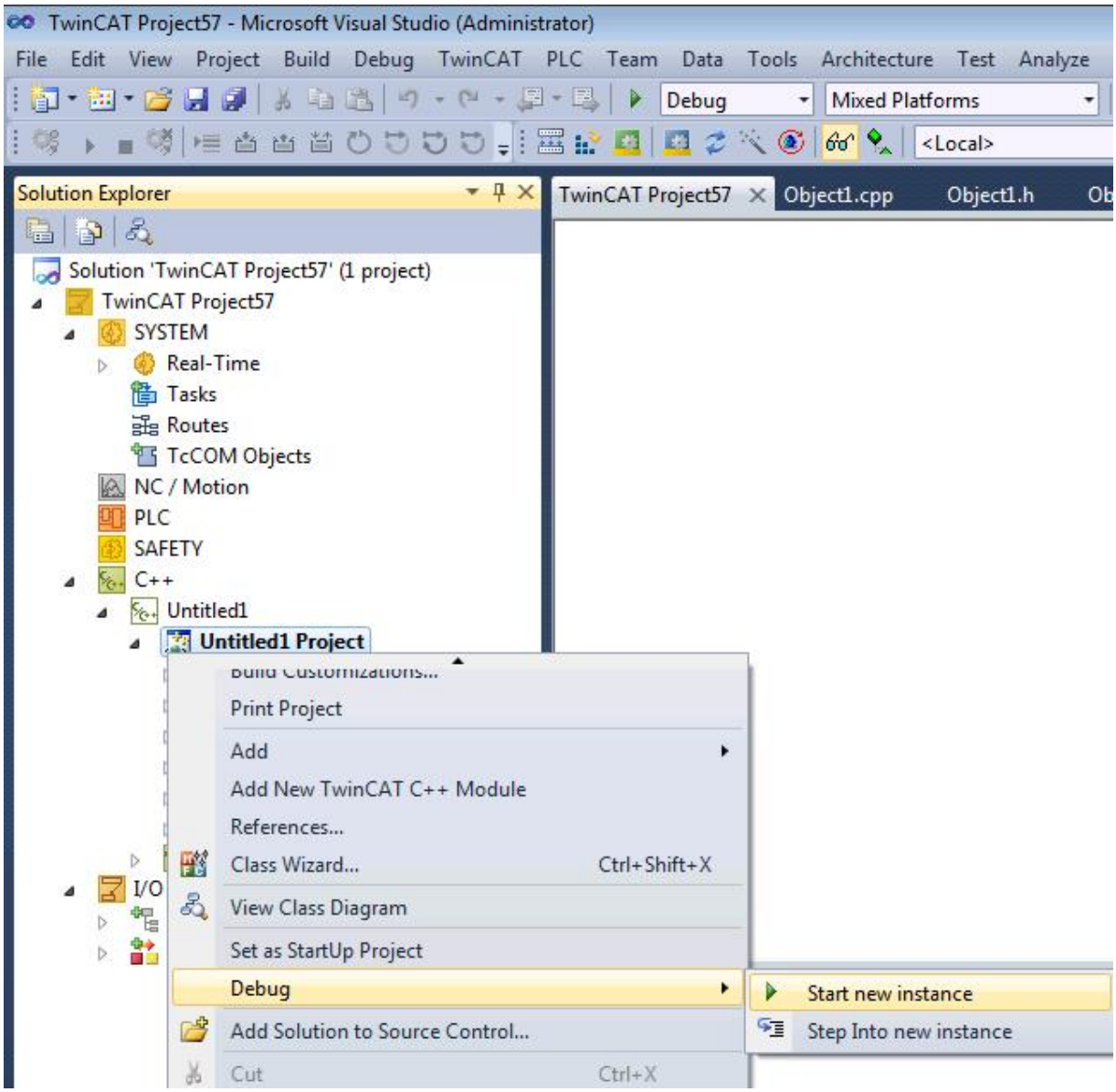
5. Wählen Sie den Zielprozess
6. Klicken Sie auf die Taste „Attach“

Option2:

Alternativ

7. Rechtsklicken Sie auf das C++ Projekt
8. Navigieren zu Debug

9. „Start new instance...“ wählen

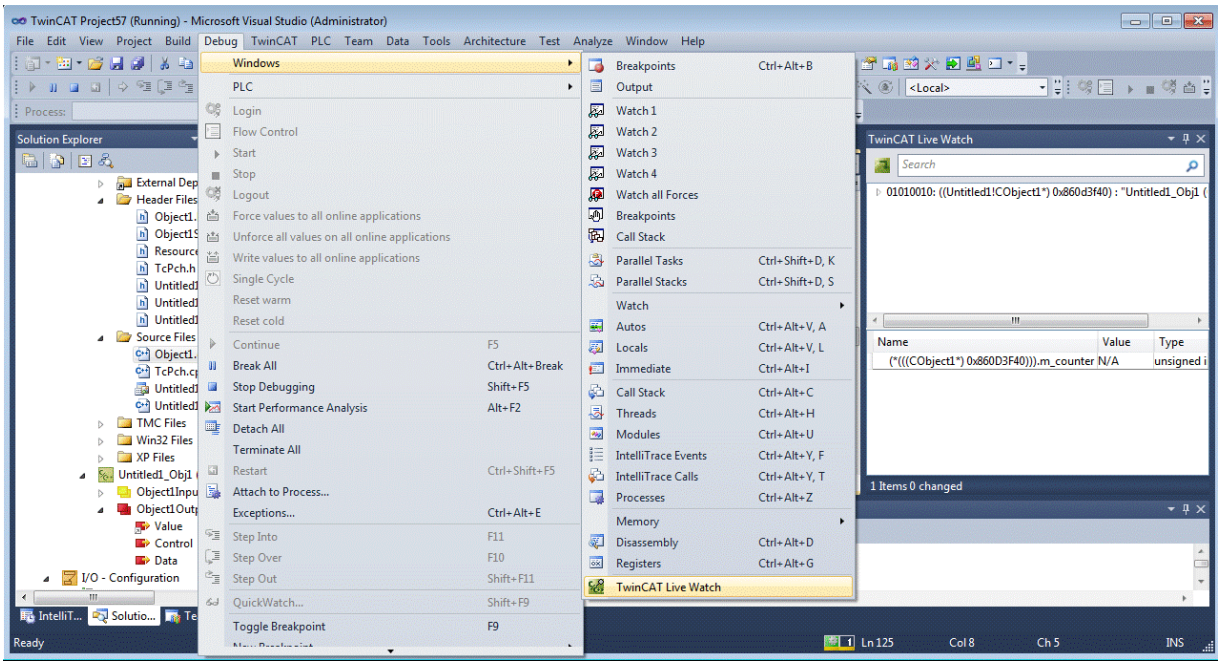


Membervariablen überwachen (ohne Haltepunkte)

Jetzt steht der „normale“ Debugging-Mechanismus von Visual Studio zur Verfügung - mit Haltepunkt setzen, schrittweise Ausführung, usw. Dessen Verwendung hängt vom zu überwachenden Prozess ab:
Wenn TwinCAT auf einer realen Maschine mit Achsenbewegungen läuft wird der Nutzer nur zur Überwachung von Variablen wahrscheinlich keinen Haltepunkt setzen wollen. Bei Erreichen eines Haltepunktes würde die Ausführung des Tasks angehalten und je nach Konfiguration die Achse sofort stoppen oder, was möglicherweise noch schlimmer wäre, sich unkontrolliert weiterbewegen - eine sehr unglückliche Situation.

TwinCAT 3 bietet die Möglichkeit, Prozessvariablen ohne das Setzen von Haltepunkten zu überwachen:

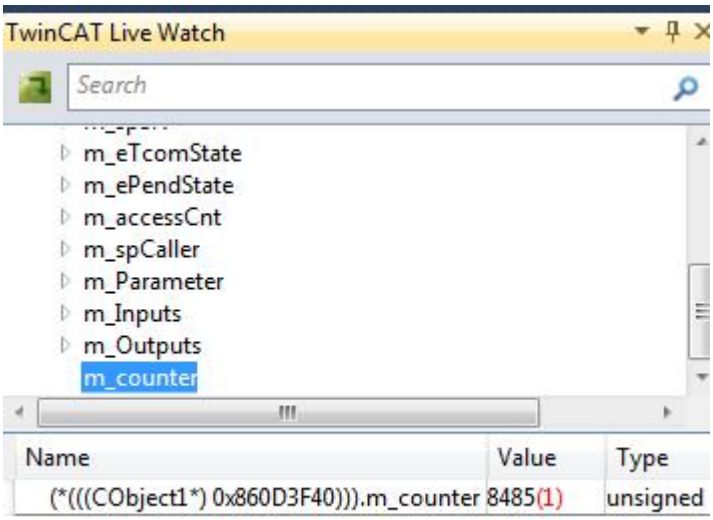
10. Wählen Sie „Debug“ -> „Windows“ -> „TwinCAT Live Watch“ für eine andere Möglichkeit, Daten zu überwachen:



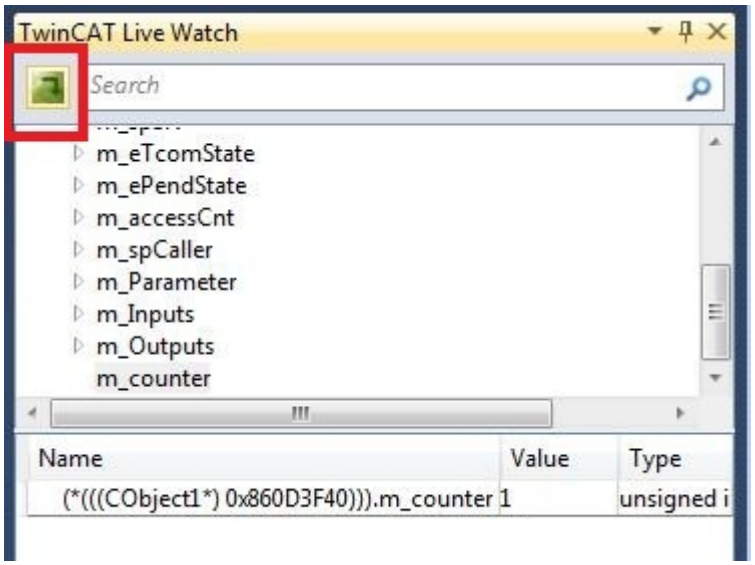
⇨ Die „TwinCAT Live Watch“-Fenster zeigen eine Liste aller Variablen im Modul. Per Drag & Drop in die Überwachungsliste (watch list) gezogene Variablen werden ohne das Setzen eines Haltepunktes überwacht.

11. Um den Wert einer Überwachungsvariablen zu verändern, einfach einen neuen Wert eingeben.

⇨ Der neue Wert wird in rot und in Klammern angezeigt.



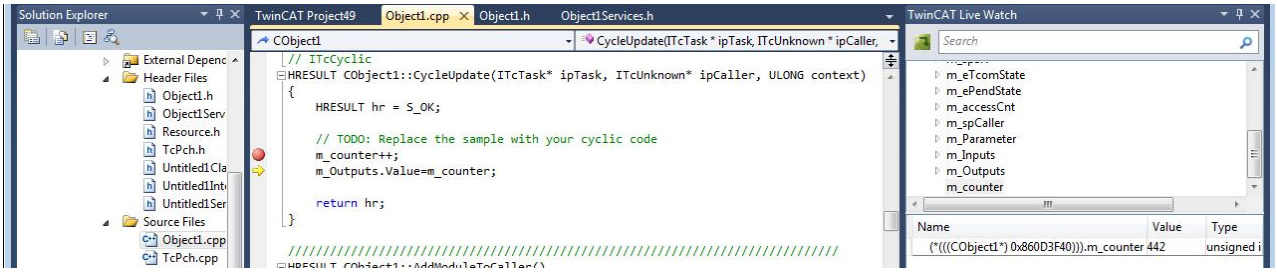
12. Durch Klicken auf das grüne Symbol wird der neue Wert in den Prozess geschrieben:



Haltepunkte setzen

Setzen von Haltepunkten auf herkömmliche Art und Weise ist ebenfalls möglich.

HINWEIS! Beachten Sie die Folgen für die Maschine!

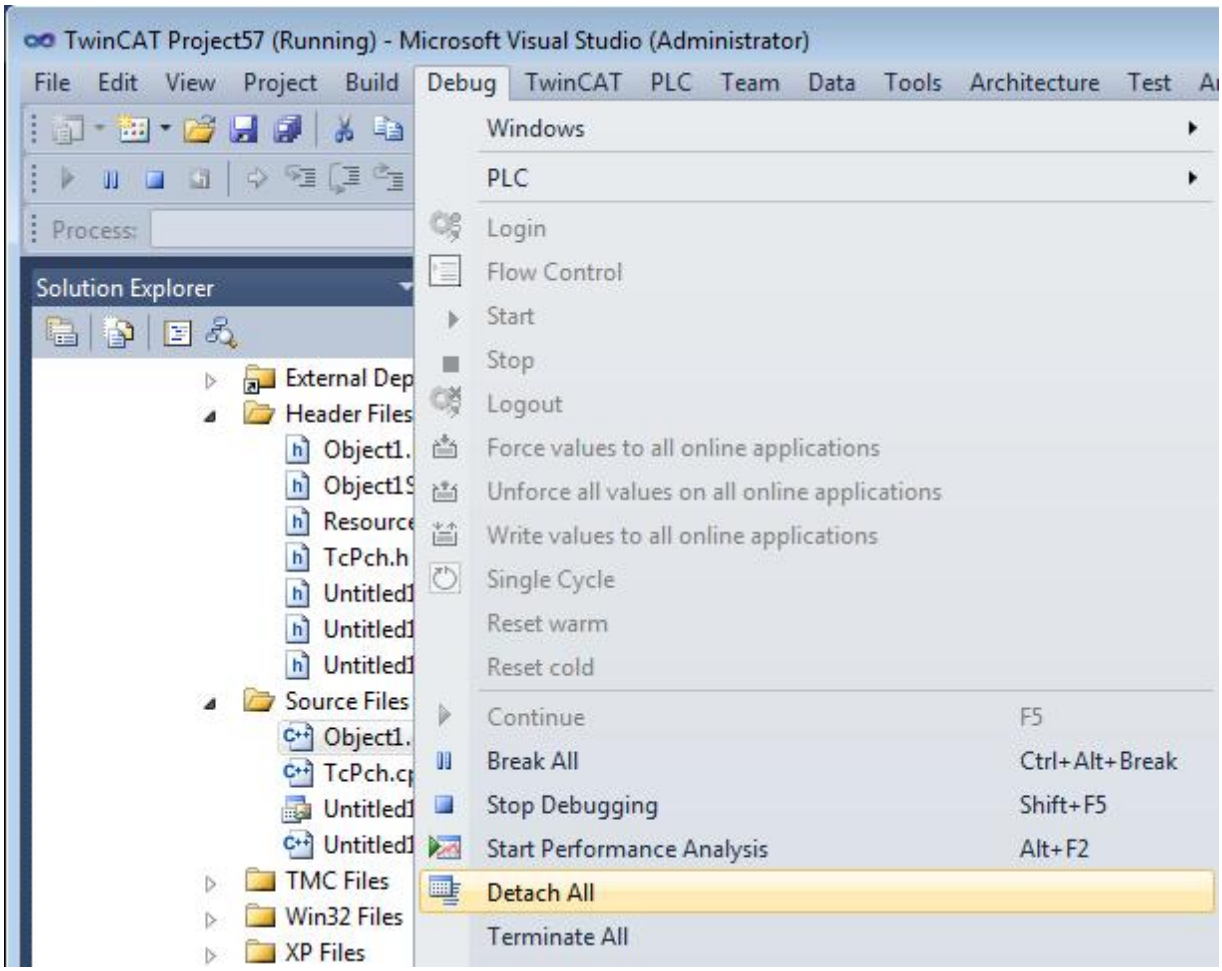


Clean up: Trennen

Es wird empfohlen, die Debug-Umgebung beim Verlassen zu bereinigen.

Debugger vom Prozess trennen

13. Klicken Sie auf „Debug“ -> „Detach All“



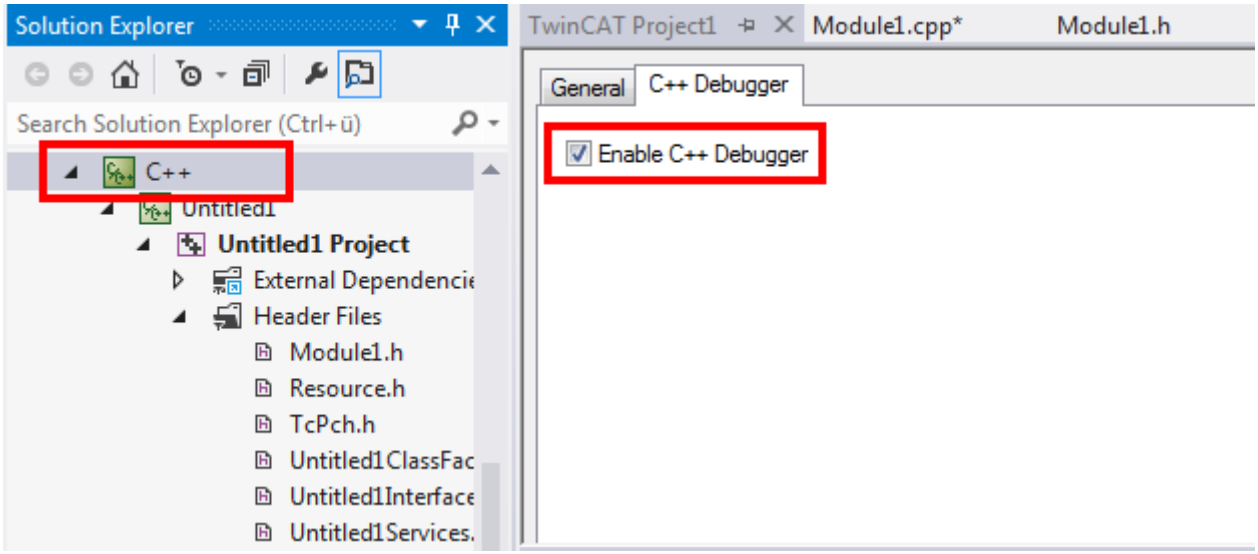
10 Debuggen

TwinCAT C++ bietet verschiedene Mechanismen für das Debuggen der unter Echtzeitbedingungen laufenden TwinCAT C++ Module.
Die meisten von ihnen entsprechen denjenigen, die von der normalen C++ Entwicklungsumgebung bekannt sind. Die Automatisierungswelt benötigt zusätzliche, leicht abweichende Debugging-Mechanismen, die hier dokumentiert sind.

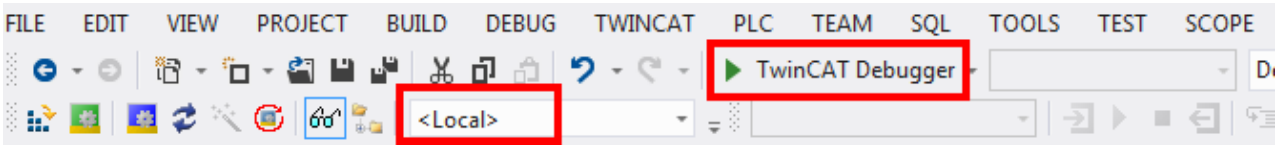
Das Debugging muss freigegeben sein.

Dies kann über den C++ Knoten der Solution konfiguriert werden:

Doppelklick auf den C++ Knoten und Wechsel zum „C++ Debugger“ Reiter, um auf das Prüfkästchen zuzugreifen.



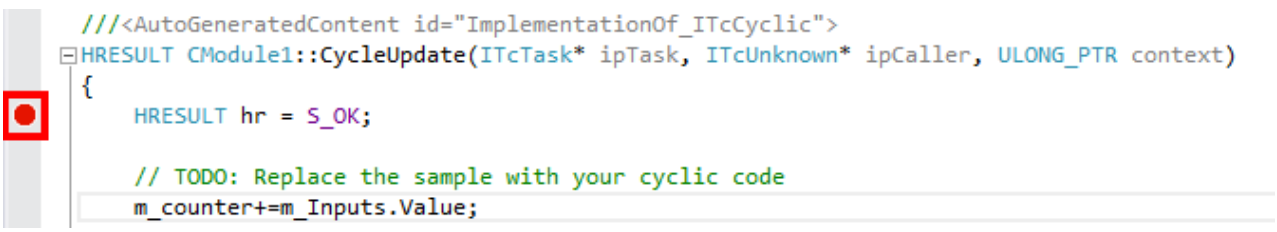
Für jegliches Debuggen in TwinCAT C++ muss das TwinCAT Engineering mit dem Laufzeitsystem (XAR) über die „TwinCAT Debugger“ Schaltfläche verbunden sein:



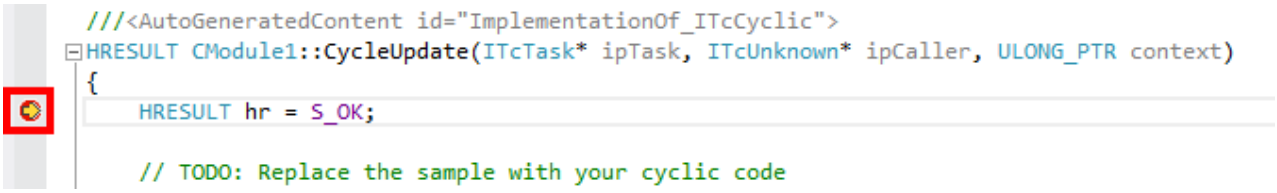
Haltepunkte und schrittweise Ausführung

In den meisten Fällen werden beim Debuggen eines C++ Programms Haltepunkte festgelegt und dann der Code schrittweise abgearbeitet, wobei die Variablen, Zeiger, usw. beobachtet werden.

TwinCAT bietet im Rahmen der Visual Studio Debugging Umgebung Möglichkeiten, um einen in Echtzeit ausgeführten Code schrittweise abzuarbeiten. Zum Festlegen eines Haltepunkts kann man durch den Code navigieren und entweder auf die graue Spalte links neben dem Code klicken oder den Hotkey (normalerweise F9) benutzen.



Beim Erreichen des Haltepunkts (wird mit Pfeil angezeigt) wird die Ausführung des Codes angehalten.

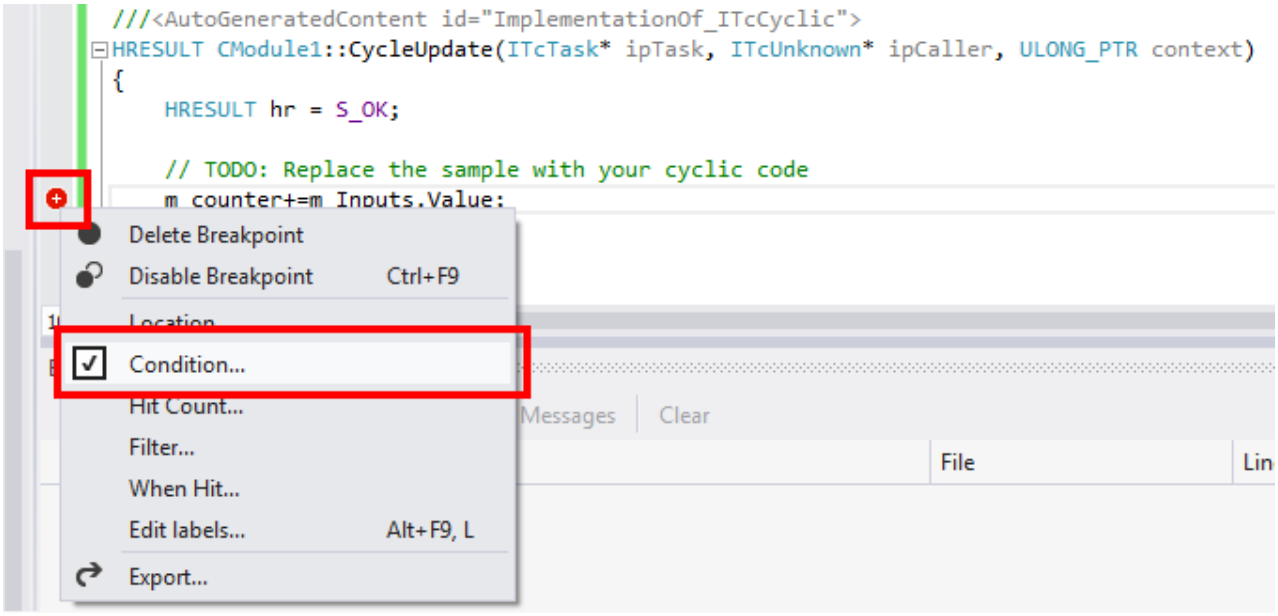


Durch Drücken auf „Step Over“ (Debug-Menü, Toolbar oder Hotkey F10) kann man den Code schrittweise ausführen. Auch stehen die von Visual Studio bekannten „Step in“ (F11) und „Step out“ (Shift+F11) zur Verfügung.

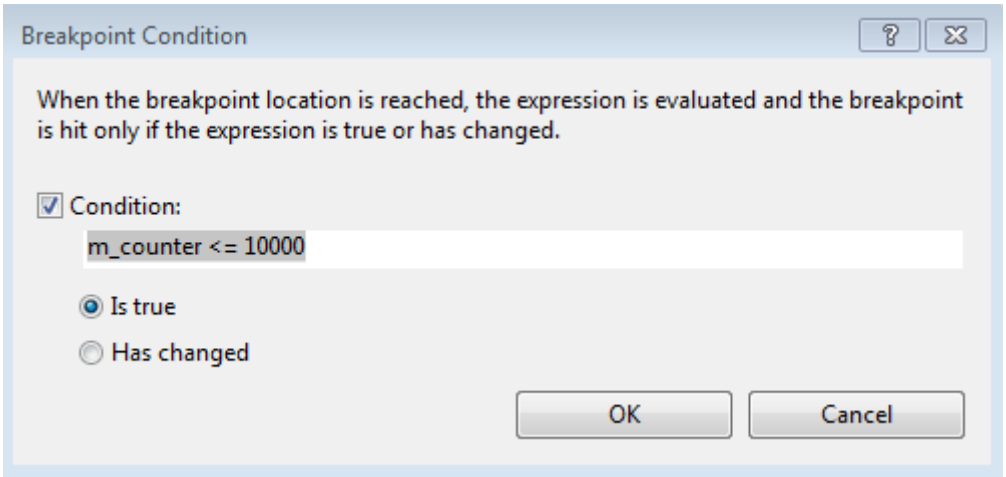
Bedingte Haltepunkte

Eine fortschrittlichere Technologie erlaubt das Setzen von bedingten Haltepunkten: Die Ausführung des Codes auf Höhe eines Haltepunktes wird nur dann angehalten, wenn eine Bedingung erfüllt ist.

TwinCAT bietet die Implementierung eines bedingten Haltepunktes als Teil der Visual Studio Integration. Zum Festlegen einer Bedingung, setzen Sie zunächst einen normalen Haltepunkt und klicken anschließend mit der rechten Maustaste auf den roten Punkt in der Haltepunkt-Spalte.



Wählen Sie „Condition…“ um das Bedingungsfenster zu öffnen:



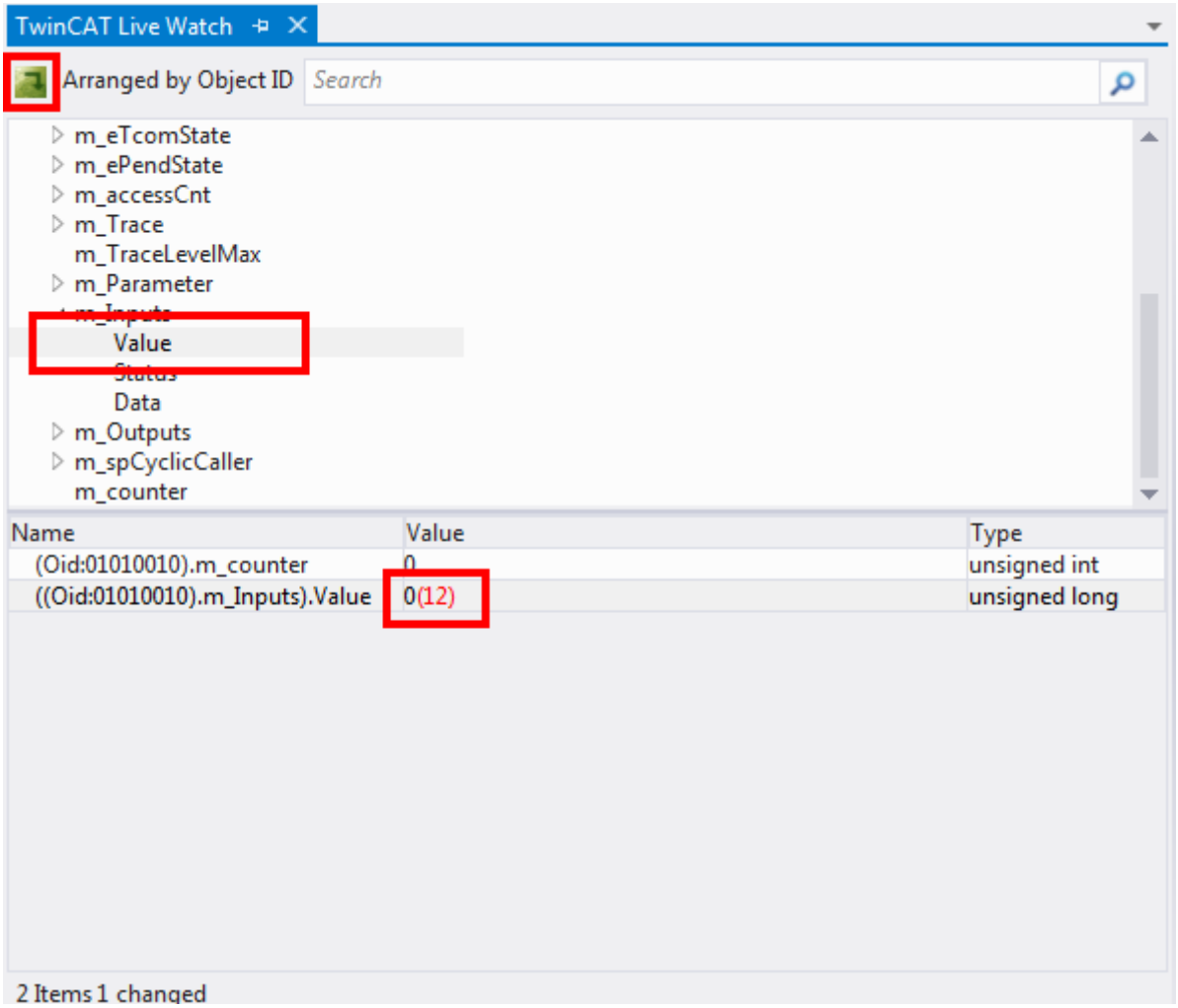
Einzelheiten zu den Bedingungen und wie diese zu formulieren sind, finden Sie hier [▶ 62].

Live Watch

Bei dem Engineering und der Entwicklung von Maschinen ist es nicht immer ratsam, das System über einen Haltepunkt anzuhalten, weil dies sich auf das Verhalten auswirken wird.
Die TwinCAT-SPS-Projekte bieten eine online-Ansicht und -Handhabung der Variablen im RUN-Zustand, ohne die Echtzeit unterbrechen zu müssen.

TwinCAT C++ Projekte bieten ein ähnliches Verhalten für C++ Code über das „Live Watch“-Fenster.

Das „Live Watch“-Fenster kann über Debug->Windows->TwinCAT Live Watch geöffnet werden.
Um das Fenster zu öffnen, zunächst eine Verbindung mit dem Echtzeitsystem herstellen (drücken der „TwinCAT Debugger“-Schaltfläche), woraufhin Visual Studio zur Debug-Ansicht wechselt, ansonsten können keine Daten bereitgestellt werden.



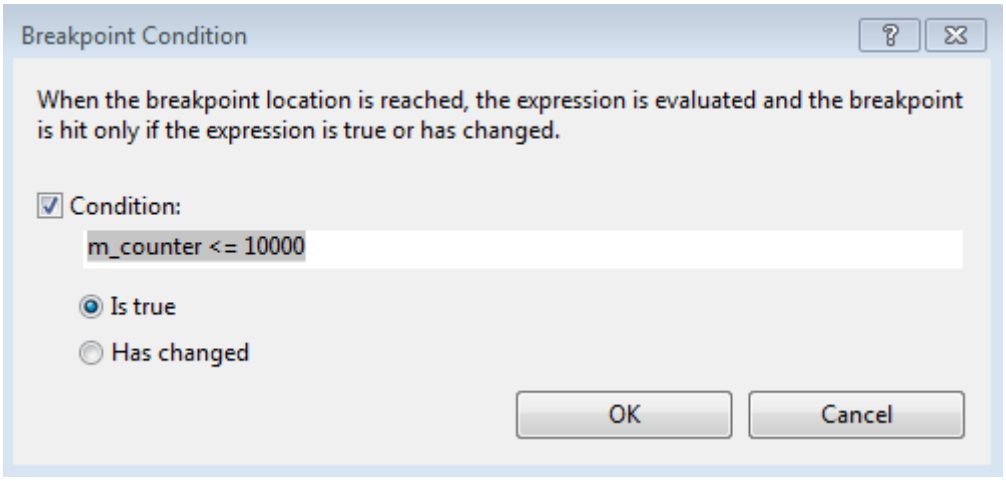
Das TwinCAT Live Watch Fenster ist in zwei Bereiche unterteilt.

Im oberen Bereich können alle Member-Variablen erkundet werden. Mittels Doppelklick auf diesen, werden sie dem unteren Bereich hinzugefügt, wo dann der aktuelle Wert angezeigt wird.

Sie können diese Werte mittels Klicken auf dem Wert im „Value“-Feld bearbeiten. Der neue Wert wird zwischen Klammern gesetzt und rot markiert. Um den Wert nun zu schreiben, drücken Sie auf das Symbol oben links.

10.1 Einzelheiten zu den bedingten Haltepunkten

TwinCAT C++ stellt bedingte Haltepunkte zur Verfügung. Einzelheiten zur Formulierung dieser Bedingungen finden Sie hier.


Breakpoint Condition
When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.
Condition:
m_counter <= 10000
Is true
Has changed
OK
Cancel

Im Gegensatz zu den Visual Studio C++ bedingten Haltepunkten werden die TwinCAT-Bedingungen kompiliert und anschließend auf das Zielsystem übertragen, sodass sie während kurzen Zyklenzeiten verwendet werden können.

Die Optionsschaltflächen bieten zwei Optionen, die getrennt voneinander beschrieben werden.

Option: Is true

Bedingungen werden mit Hilfe von logischen Termen, vergleichbar mit den konjunktiven Normalformen definiert.
Sie werden aus einer Kombination von mit && verbundenen „Maxterms“ gebildet:

```
(Maxterm1 && Maxterm2 && ... && MaxtermN)
```

wobei jedes Maxterm eine Kombination von || verbundenen Bedingungen darstellt:

```
(condition1 ||condition2 || ... || conditionN )
```

Mögliche Vergleichsoperatoren: `==, !=, <=, >=, <, >`

Für die Bestimmung der verfügbaren Variablen siehe Live Watch Fenster. Alle aufgeführten Variablen können für die Formulierung von Bedingungen herangezogen werden. Dazu gehören sowohl TMC-definierte Symbole, als auch lokale Member-Variablen.

Beispiele:

```
m_counter == 123 && hr != 0
```

```
m_counter == 123 || m_counter2 == 321 && hr == 0
```

```
m_counter == 123
```

Weitere Anmerkungen:

- Überwachung von Modul-Instanzen:
  Die OID des Objekts ist in `m_objId` gespeichert, somit kann z.B. die Überwachung des OID folgendermaßen aussehen `m_objId == 0x01010010`
- Überwachung von Tasks:
  Es wird eine spezielle Variable `#taskId` bereitgestellt, um auf die OID des aufrufenden Tasks zugreifen zu können. Z.B. `#taskID == 0x02010010`

Option: Has changed

Die Option „Has changed“ ist einfach zu verstehen: Indem Variablennamen bereitgestellt werden, wird der Wert überwacht und die Ausführung angehalten, wenn der Wert sich gegenüber dem vorangegangenen Zyklus geändert hat.

Beispiele:

```
m_counter
```

```
m_counter && m_counter2
```

11 Assistenten

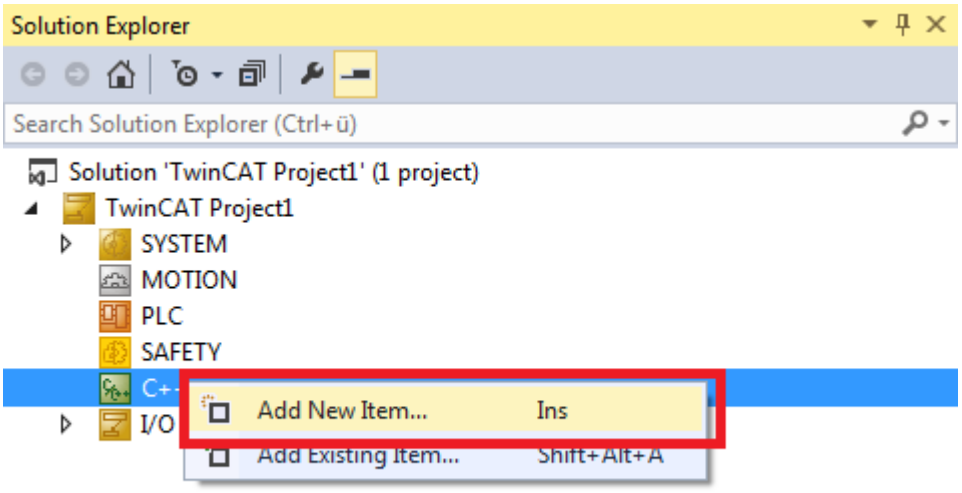
Um den Einstieg in das Engineering des TwinCAT C++ Systems zu vereinfachen, stehen Assistenten zur Verfügung.

- Der TwinCAT Projekt-Assistent [▶ 65] erstellt ein TwinCAT C++ Projekt. Im Falle von Treiber-Projekten wird anschließend der TwinCAT Class Wizard gestartet.
- Der TwinCAT Module Class Wizard [▶ 66] wird automatisch bei der Erstellung eines C++ Moduls gestartet.
  Dieser Assistent stellt verschiedene „gebrauchsfertige“ Projekte als Einstiegspunkt für eigene Entwicklungen zur Verfügung.
- Der TwinCAT Module Class Editor [▶ 69] (TMC) ist ein grafischer Editor für die Definition von Datenstrukturen, Parametern, Datenbereichen, Schnittstellen und Zeigern. Er erzeugt eine TMC-Datei, die vom TMC-Code-Generator verwendet wird.
- Anhand der definierten Klassen werden Instanzen generiert, die über den TwinCAT Module Instance Configurator [▶ 112] konfiguriert werden können.

11.1 TwinCAT C++ Projekt-Assistent

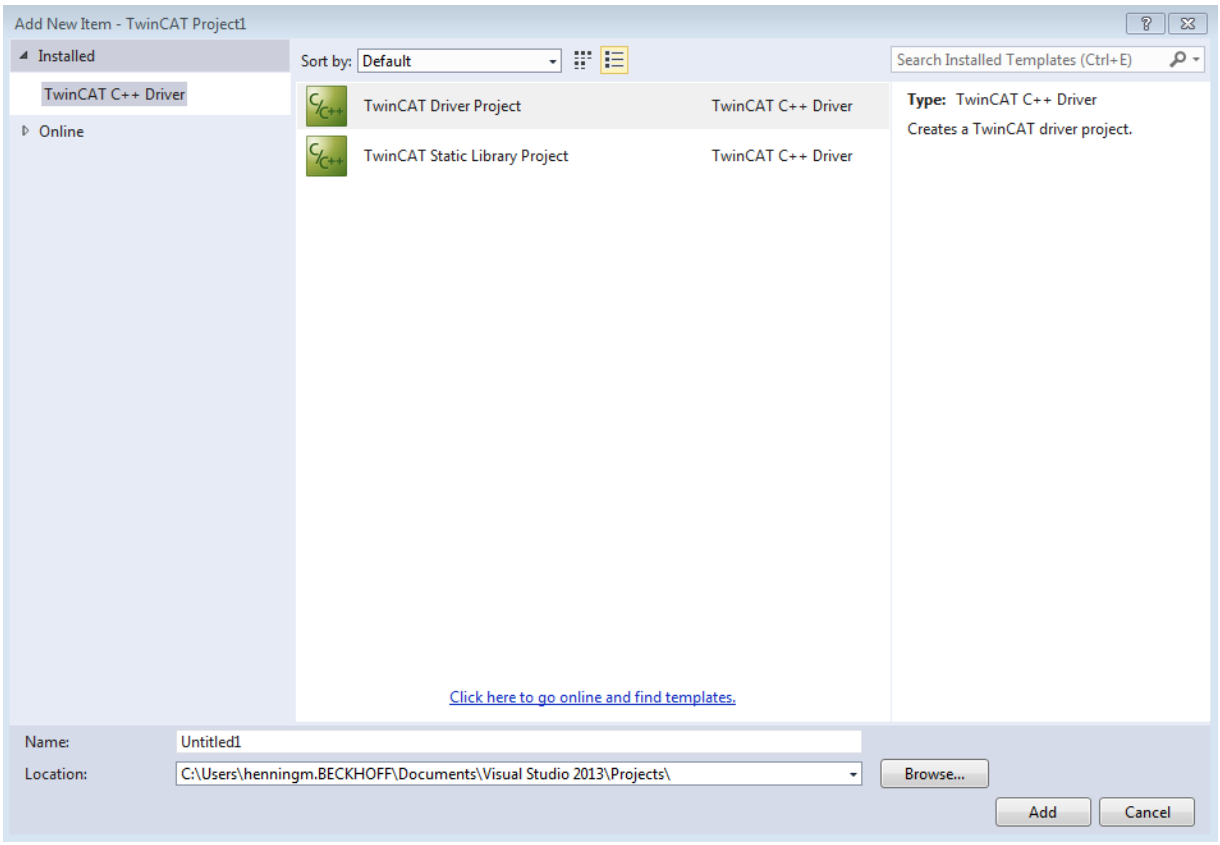
Nach der Erstellung eines TwinCAT Projekts, kann man mit Hilfe des TwinCAT C++ Projekt-Assistenten ein C++ Projekt hinzufügen:

1. Rechtsklicken auf C++ Symbol und „Add new Item…“ auswählen, um den C++ Projektassistenten zu starten.



TwinCAT bietet zwei C++ Projekte an:
Treiber-Projekt: Projekte, die ein oder mehrere ausführbare Module beinhalten

Statische Bibliothek: Projekte mit C++ Funktionen, die von (verschiedenen) TwinCAT C++ Treibern verwendet werden.



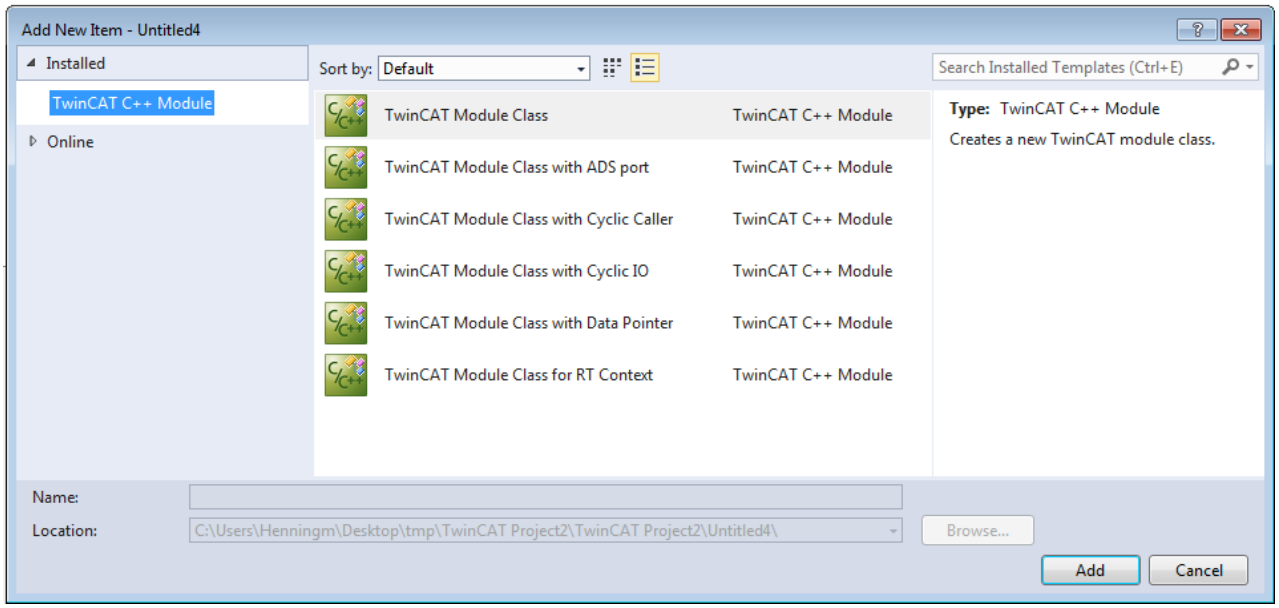
2. Wählen Sie eine der Projektvorlagen, geben einen Namen und einen Speicherort an.
   ⇨ Das TwinCAT C++ Projekt wird erstellt

⇨ Im Falle eines Treibers wird der TwinCAT C++ Klassenassistent [▶ 66] gestartet.

11.2 TwinCAT Module Klassenassistent

TwinCAT 3 bietet verschiedene Klassenvorlagen

- TwinCAT Modules Class
- TwinCAT Modules Class mit ADS Port
- TwinCAT Modules Class mit zyklischem Aufrufer
- TwinCAT Modules Class mit zyklischem Ein-/Ausgang
- TwinCAT Modules Class mit Datenzeiger
- TwinCAT Modules Class für Echtzeitkontext



TwinCAT Modules Class

Erstellt eine neue TwinCAT-Modulklasse.

Diese Vorlage erzeugt ein grundlegendes Kernmodul. Es verfügt weder über einen zyklischen Aufrufer, noch über einen Datenbereich, ist aber ein guter Ausgangspunkt für die Implementierung von abrufbaren Funktionen auf Anfrage eines Aufrufers.

Zum Beispiel für die Erstellung einer C++ Methode, die von einem SPS-Modul oder einem anderen C++ Modul aufgerufen wird.

Siehe Beispiel11 [▶ 217]

TwinCAT Modules Class mit ADS Port

Diese Vorlage bietet sowohl das C++ Modul als auch die Funktionsweise eines ADS Servers und ADS Clients.

- ADS Server:
  Kann als einzelne Instanz dieser Vorlage des C++ Moduls laufen und kann mit einer spezifischen ADS Port-Nummer (z.B. 25023) vorkonfiguriert werden.
  Ermöglicht mehrere Instanzen dieser Vorlage, wobei jedem C++ Modul seine eigene eindeutige ADS Port-Nummer von TwinCAT 3 zugewiesen wird (z.B. 25023, 25024, 25025, ...).
  Die ADS-Meldungen können dank der Implementierung des C++ Moduls analysiert und verarbeitet werden.
  Das ADS Handling zwecks Zugriff auf Ein-/Ausgangsdatenbereiche muss nicht über ein eigenes ADS Message Handling implementiert werden.
- ADS Client:
  Diese Vorlage stellt Beispielcodes zur Verfügung, um einen ADS-Aufruf mittels Versand einer ADS-Meldung an einen ADS-Partner zu initiieren.

Da die Module sich wie ADS Client oder ADS Server verhalten, die untereinander über ADS-Nachrichten kommunizieren, können die beiden Module (Aufrufer=Client und der Aufgerufene=Server) im gleichen oder unterschiedlichen Echtzeitkontexten auf dem gleichen oder unterschiedlichen CPU-Kernen laufen.
Weil ADS netzübergreifend arbeiten kann, können die beiden Module auch auf verschiedenen im Netzwerk befindlichen Maschinen laufen...

Siehe Beispiel03 [▶ 200], ADS-Kommunikation [▶ 155]

TwinCAT Modules Class mit zyklischem Aufrufer

Es ermöglicht den zyklischen Aufruf eines C++ Programms, das aber über keinen Zugang zur Außenwelt verfügt.

Dieses wird nicht häufig verwendet. Eine Modulklasse mit zyklischem Aufrufer und zyklischem I/O wird bevorzugt.

TwinCAT Modules Class mit zyklischem Ein-/Ausgang

Erzeugt eine neue TwinCAT-Modulklasse, die die zyklisch aufrufende Schnittstelle implementiert und die einen Ein- und Ausgangsdatenbereich aufweist.

Die Ein- und Ausgangsdatenbereiche können mit anderen Ein-/Ausgangsabbildern oder mit physikalischen E/A-Klemmen verbunden werden.

Wichtig zu verstehen:

Das C++ Modul verfügt über seinen eigenen logischen Ein-/Ausgangsdatenspeicherbereich. Die Datenbereiche des Moduls können mit dem Systemmanager konfiguriert werden.

Wenn das Modul mit einer zyklischen Schnittstelle gemappt wird, bestehen Kopien der Ein- und Ausgangsdatenbereiche in beiden Modulen (dem Aufrufer und dem Aufgerufenen) und auf diese Weise kann das Modul unter einem anderen Echtzeitkontext und selbst auf einem anderen CPU-Kern in Bezug auf ein anderes Modul laufen.
TwinCAT wird die Daten zwischen den Modulen auf ununterbrochene Weise kopieren.

Siehe Schnellstart [▶ 37], Beispiel01 [▶ 198]

TwinCAT Modules Class mit Datenzeiger

Genau wie das „TwinCAT Module Class with Cyclic IO“ erzeugt auch diese Vorlage eine neue TwinCAT-Modulklasse die eine aufrufende Schnittstelle mit einem Ein- und Ausgangsdatenbereich für die Verknüpfung mit anderen Logik-Ein-/Ausgangsabbildern oder mit physikalischen E/A-Klemmen implementiert.

Darüber hinaus bietet diese Vorlage „Datenzeiger“, mit Hilfe derer auf Datenbereiche von anderen Modulen über Zeiger zugegriffen werden kann.

Wichtig zu verstehen:

Anders als im Falle des zyklischen I/O-Datenbereichs, wo die Daten zwischen Modulen zyklisch kopiert werden, besteht im Falle der Verwendung von C++ „Datenzeigern“ nur ein einziger Datenbereich und dieser gehört dem Zielmodul. Beim Schreiben von einem anderen C++ Modul über den „Datenzeigermechanismus“ auf das Zielmodul wird sich das sofort auf den Datenbereich des Zielmoduls auswirken. (Nicht notwendigerweise gegen Ende eines Zyklus).

Wenn das Modul während der Laufzeit ausgeführt wird, findet der Aufruf sofort statt, wodurch der ursprüngliche Prozess blockiert wird (es ist ein Zeiger...). Aufgrund dessen müssen beide Module (der Aufrufer und der Aufgerufene) sich im **selben Echtzeitkontext** und auf demselben CPU-Kern befinden.

Der „Datenzeiger“ wird im TwinCAT Module Instance Configurator [▶ 112] konfiguriert.

Siehe Beispiel10 [▶ 216]

TwinCAT Modules Class für Echtzeitkontext

Diese Vorlage erstellt ein Modul, das unter Echtzeitbedingungen instanziiert werden kann.

Wie hier [▶ 27] beschrieben, verfügen andere Module über Übergänge für Ein-/Ausschalten in Nicht-Echtzeitkontext. Manchmal müssen die Module bei bereits laufender Echtzeit gestartet werden, also müssen alle Übergänge unter Echtzeitbedingungen ausgeführt werden. Dies ist eine entsprechende Vorlage.

11.3 TwinCAT Module Class Editor (TMC)

Der TwinCAT Module Class Editor (kurz TMC Editor) wird für die Definition der Klasseninformation eines Moduls verwendet. Es handelt sich um Datentypdefinitionen mit deren Verwendung, bereitgestellte und implementierte Schnittstellen, sowie Datenbereiche und Datenzeiger.

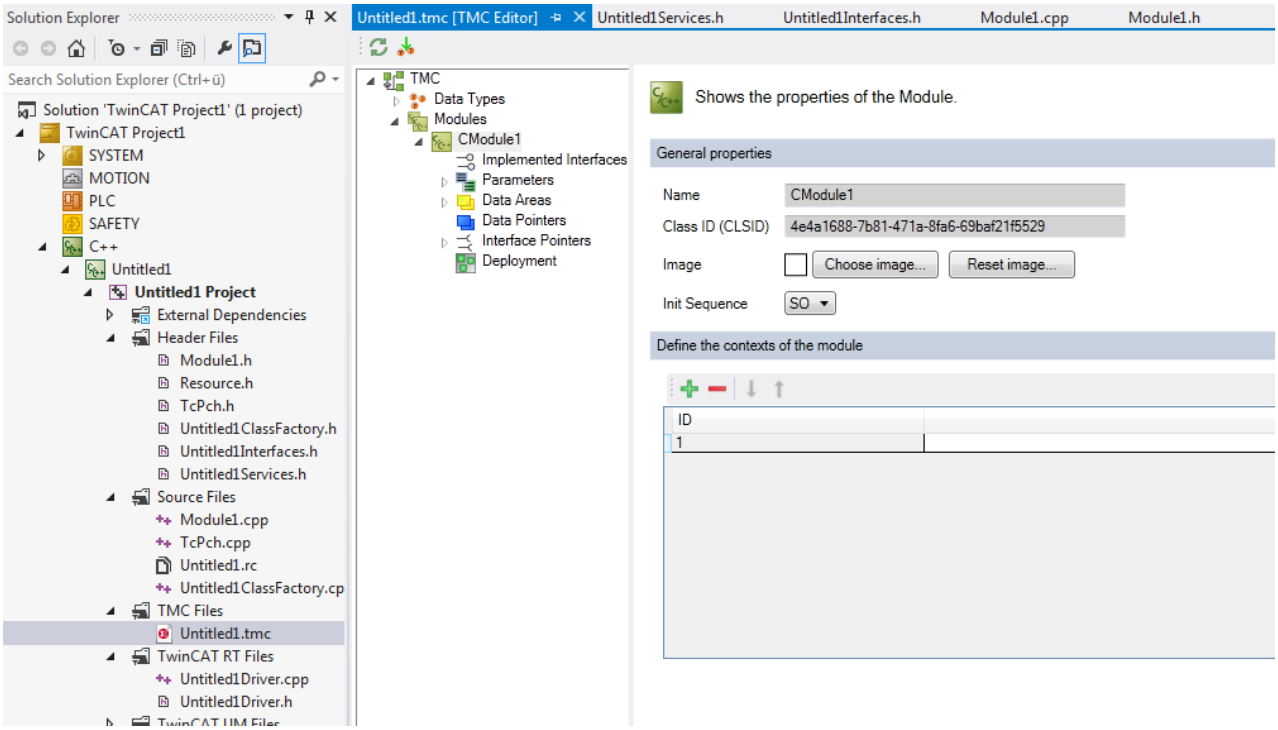
Kurz: Alles was von außen zu sehen ist, muss mit diesem Editor definiert werden.

Die Grundidee ist:

1. TwinCAT Module Class Editor (TMC) [▶ 69] und verwenden Sie ihn, um einfach Funktionalitäten in die TwinCAT C++ TMC-Datei (das ist die XML basierte Beschreibung der TwinCAT C++ Modulklasse) hinzuzufügen / zu ändern.
2. TwinCAT Module Class Editor (TMC) [▶ 70] kann der gesamte notwendige C++ Code, d.h. Header- und cpp-Dateien, erzeugt werden.

Starten Sie den TMC Editor

Öffnen Sie den Editor mittels Doppelklick auf die TMC-Datei eines Moduls. Der grafische Editor wird geöffnet:

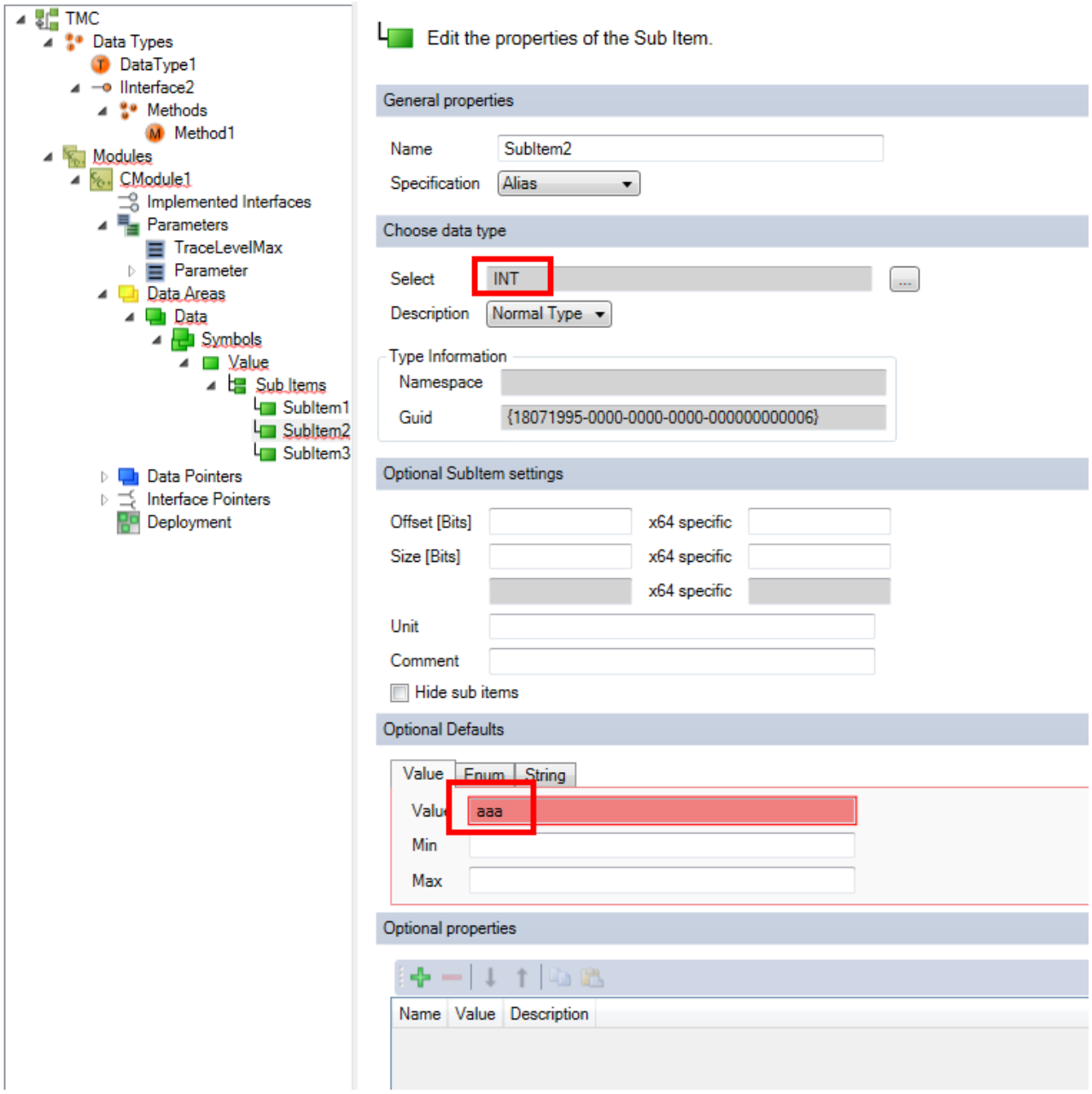


Funktionalitäten des TMC Editors sind:

- Symbole in den Datenbereichen, z.B. wie die logischen Ein- oder Ausgangsprozessabbilder eines Moduls erstellen / löschen / bearbeiten
- Benutzerdefinierte Datentypdefinitionen erstellen / löschen / bearbeiten
- Symbole in der Parameterliste eines Moduls erstellen / löschen / bearbeiten

Der TMC Editor unterstützt den Kunden bei der Definition seiner Datentypen und C++ Module.

Zum Beispiel bei Problemen (Alignment, ungültige Standarddefinitionen, ...) innerhalb des TMC, wird der Nutzer über die Anzeige von roten Markierungen innerhalb des TMC-Baums zur entsprechenden Stelle geführt:

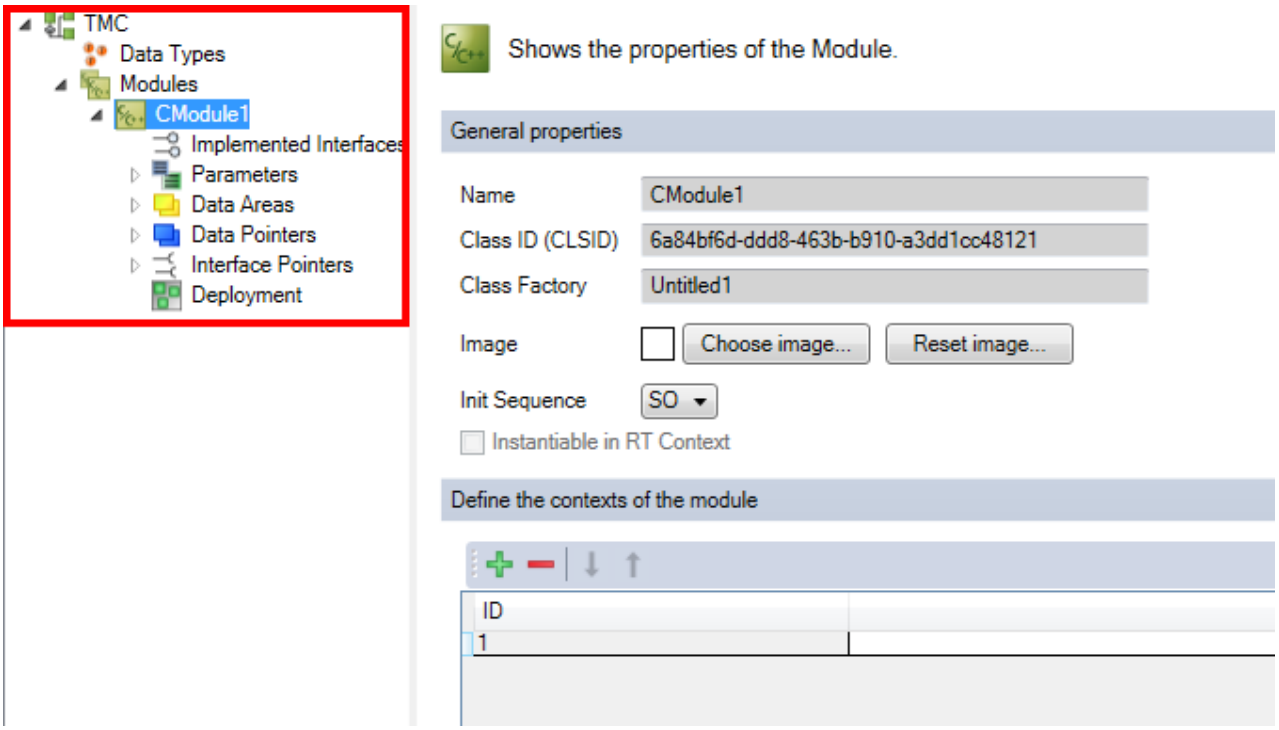


Nichtsdestotrotz kann der Nutzer direkt die TMCs bearbeiten, weil es sich um XML handelt und somit vom Nutzer selber erzeugt und bearbeitet werden kann.

Den TMC Code Generator ausführen

Der Editor wird die eingegebene Information in die TMC-Datei speichern. Diese TMC-Beschreibung wird vom „TwinCAT TMC Code Generator“ in Quellcode umgewandelt, der dann im Kontextmenü des TwinCAT C ++ Projekts verfügbar ist.

11.3.1 Übersicht



Benutzerinterface

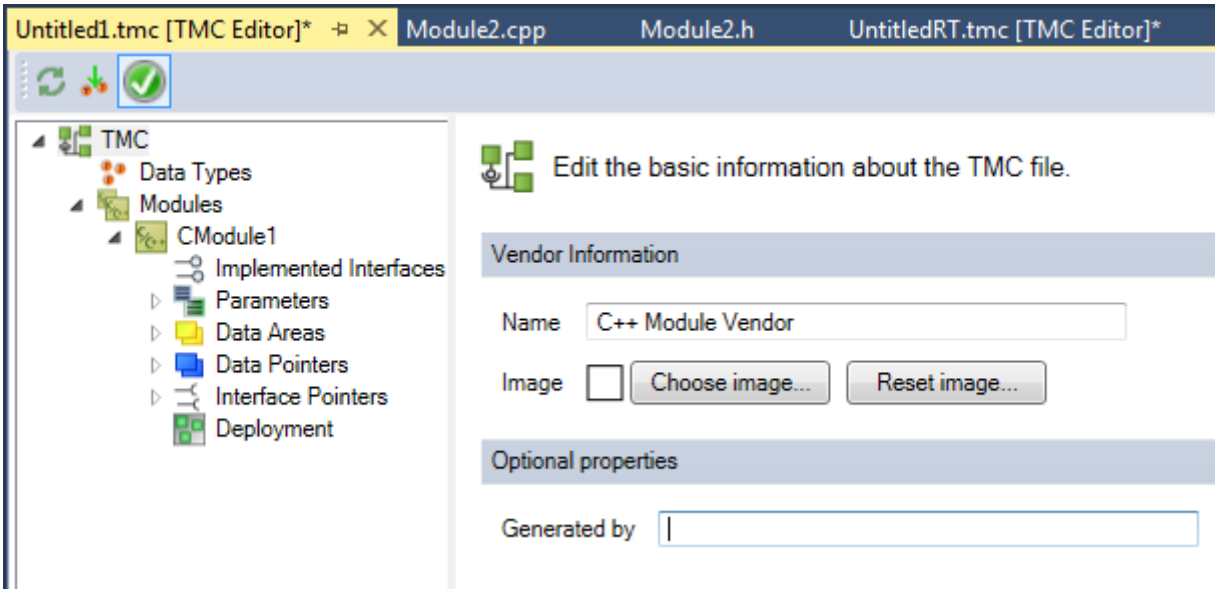
- TMC [▶ 71]: Die grundlegenden Angaben des Herstellers des C++ Moduls bearbeiten und ein Bild hinzufügen
- Data Types [▶ 73]: Datentypen hinzufügen, entfernen und neu ordnen
- Modules [▶ 90]: Zeigt die Module des Treibers
- Implemented Interfaces [▶ 93]: Zeigt die implementierten Schnittstellen des Moduls.
- Parameters [▶ 93]: Ihre Parameter hinzufügen, entfernen und neu ordnen
    - TraceLevelMax [▶ 100]: Parameter, der die Menge an protokollierten Nachrichten steuert - für (fast) jedes Modul vordefiniert.
- Data Areas [▶ 101]: Datenbereiche hinzufügen, entfernen und neu ordnen
- Data Pointers [▶ 108]: Datenzeiger hinzufügen, entfernen und neu ordnen
- Interface Pointers [▶ 110]: Datenzeiger hinzufügen, entfernen und neu ordnen
- Deployment [▶ 111]: Bestimmt die Dateien, die bereitgestellt werden

Sehen Sie dazu auch

- Datentypen hinzufügen / bearbeiten / löschen [▶ 74]

11.3.2 Grundlegende Informationen

Basic information: Grundlegende Informationen bezüglich der TMC-Datei bearbeiten



Informationen zum Anbieter

Name: Modulnamen bearbeiten

Choose Image: Ein 16x16 Pixel-Bitmap-Symbol einfügen

Reset image: Modulbild auf Standardwert zurücksetzen

Optionale Eigenschaften

Generated by: In diesem Feld wird angegeben, wer die Datei erstellt hat und wer diese pflegen wird. Beachten Sie, dass beim Ausfüllen dieses Felds Änderungen nicht mehr möglich sind (deaktiviert alle Bearbeitungsvorgänge) im TMC Editor.

11.3.3 Datentypen

Im TwinCAT Module Class (TMC) Editor können benutzerdefinierte Datentypen definiert werden.

Bei diesen Datentypen kann es sich um Typendefinitionen, Strukturen, Bereiche, Aufzählungen oder Schnittstellen, z.B. Methoden und deren Signaturen, handeln.

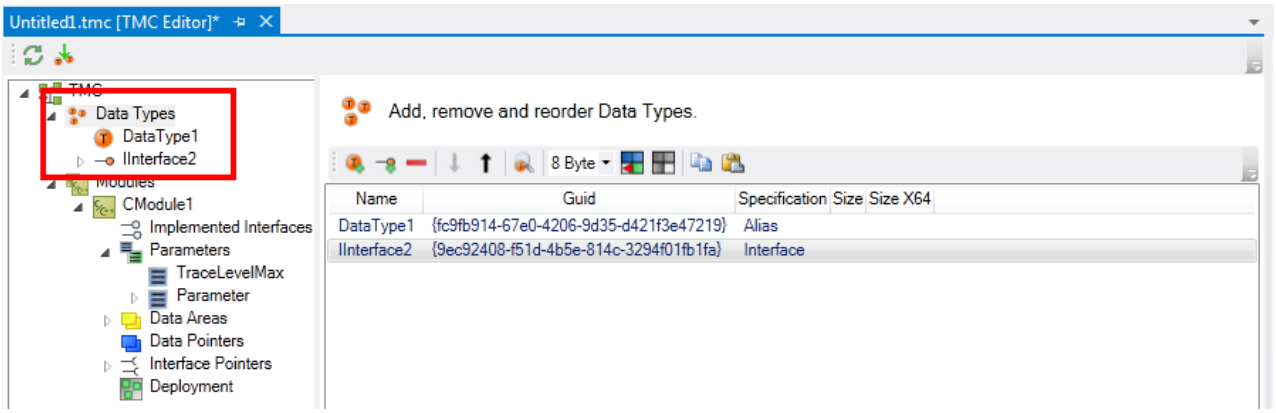
Das TwinCAT Engineering System (XAE) veröffentlicht diese Datentypen gegenüber allen anderen verschachtelten Projekten des TwinCAT Projekts, sodass diese auch z.B. in SPS-Projekten verwendet werden können (wie hier [▶ 217] beschrieben).

Hinweis	**Namenskonflikt** Bitte verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen. Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.

In diesem Kapitel wird beschrieben, wie die Fähigkeiten des TMC Editors bei der Definition von Datentypen zu verwenden sind.

11.3.3.1 Übersicht

Benutzerinterface



Symbol	Funktion
	Einen neuen Datentyp hinzufügen
	Eine neue Schnittstelle hinzufügen
	Löscht den ausgewählten Typ
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben
	Sucht nicht verwendete Typen
	Byte Alignment auswählen
	Ausgewählten Datentyp ausrichten
	Datenformat des ausgewählten Datentyps zurücksetzen
	Kopieren
	Einfügen

Datentypeigenschaften

Name: Benutzerdefinierte Name des Datentyps

GUID: Eindeutige ID des Datentyps

Specification: Festlegung des Datentyps

Size: Größe des Datentyps, wenn ausdrücklich spezifiziert

Size X64: Unterschiedliche Größe des Datentyps für x64-Plattform

11.3.3.2 Datentypen hinzufügen / bearbeiten / löschen

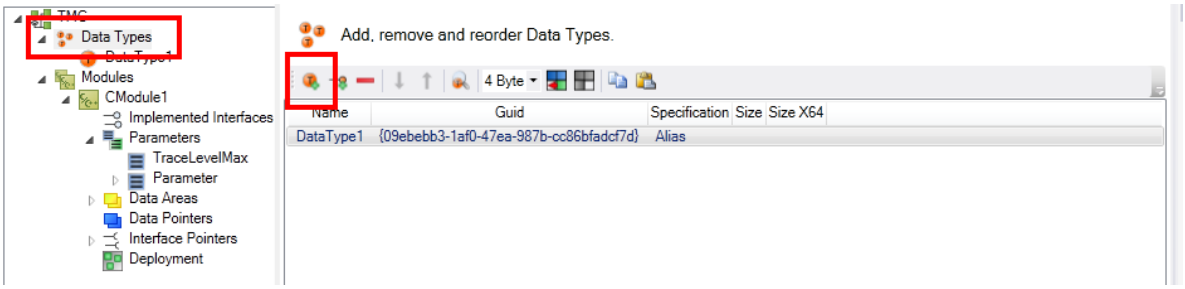
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Datentypen, die von TwinCAT C++ Modulen verwendet werden, hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

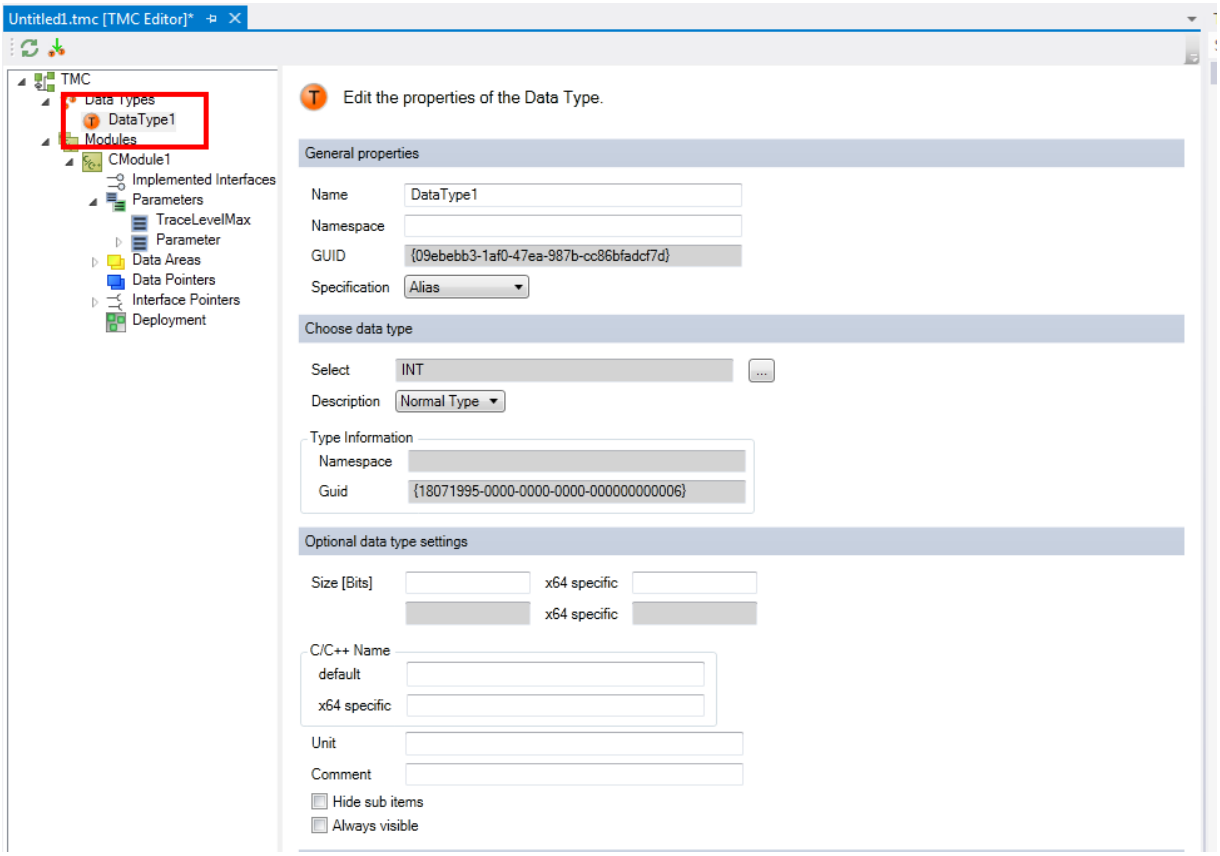
- Schritt 1: Einen neuen Datentyp [▶ 74] in der TMC-Datei erstellen
- Schritt 2: TwinCAT TMC Code Generator starten [▶ 77], um C++ Code auf der Grundlage einer Modulbeschreibung in der TMC-Datei zu generieren
- Die Datentypen verwenden [▶ 90]

Schritt 1: Einen neuen Datentyp erzeugen

1. Nach dem Starten des TMC Editors den Knoten **„Data Types“** auswählen.
2. Die Liste der Datentypen und Schnittstellen wird mit einem neuen Datentyp durch Klicken auf die „+“ Schaltfläche „Add a new data area“ erweitert.
   ⇨ Daraufhin wird ein neuer **„Datentyp“** als neuer Eintrag aufgeführt:

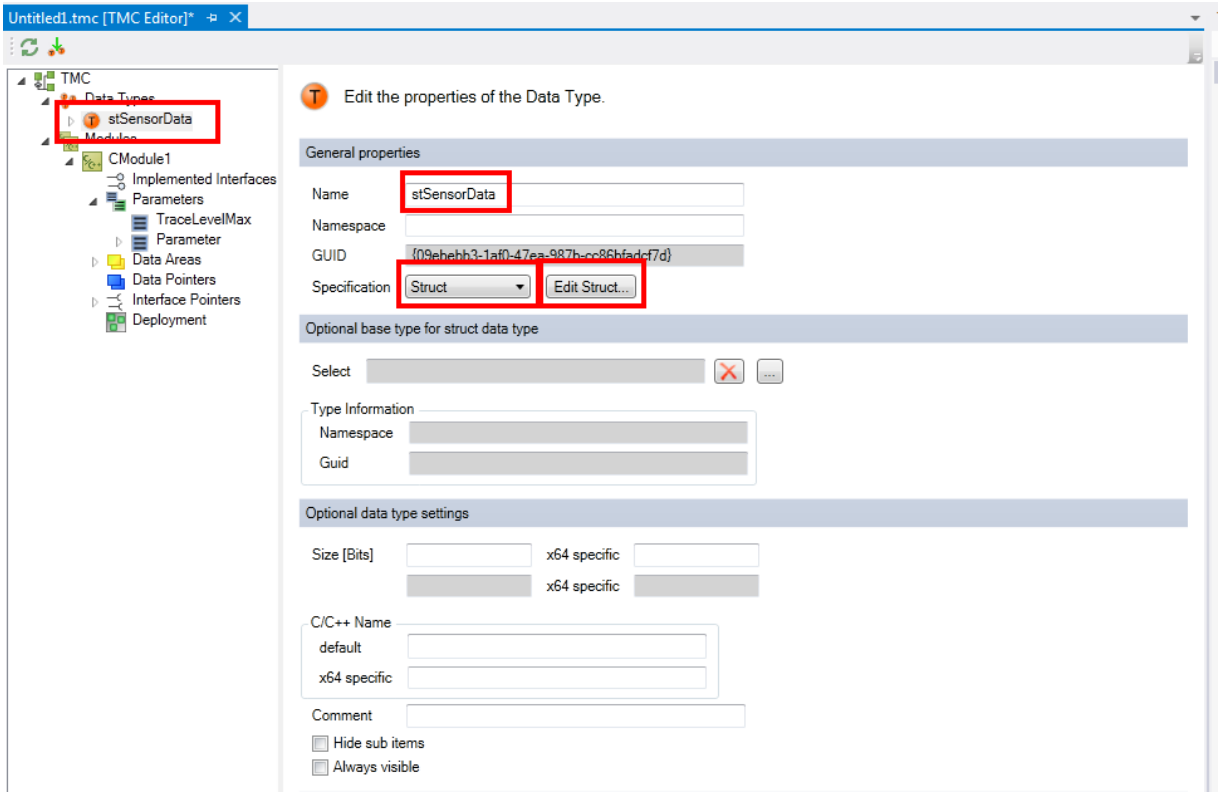


3. Wählen Sie den generierten „Data Type1“ um Einzelheiten zum neuen Datentyp zu erhalten.

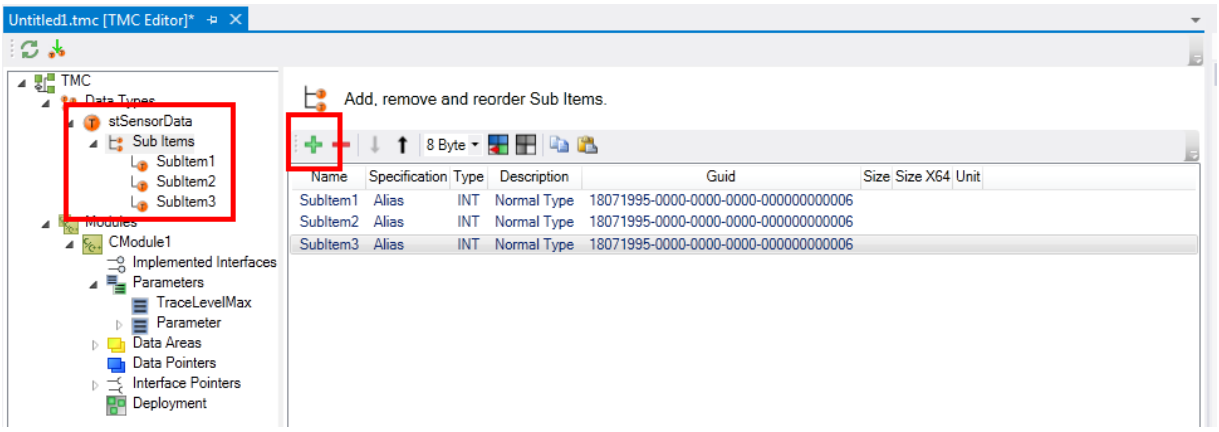


4. Den Datentyp spezifizieren.
   Genaueres siehe hier [▶ 84].

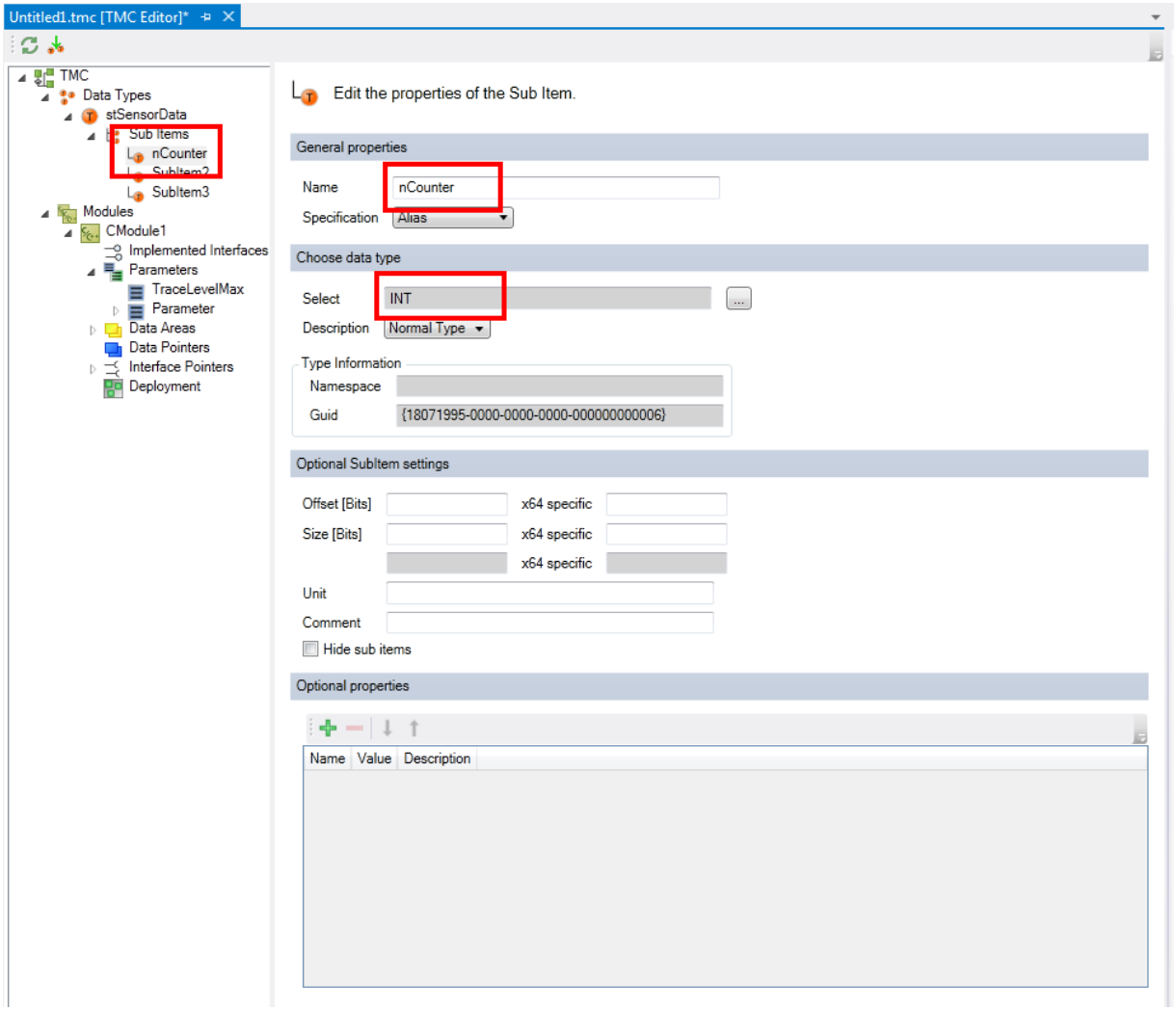
5. Den Datentyp umbenennen.
   In diesem Beispiel **„stSensorData“** wählen Sie die Spezifikation „STRUCT“ und klicken auf „Edit Struct“.



6. Fügen Sie neue Unterelemente in die Struktur ein mittels Klicken auf die „Add a new sub item“-Schaltfläche.



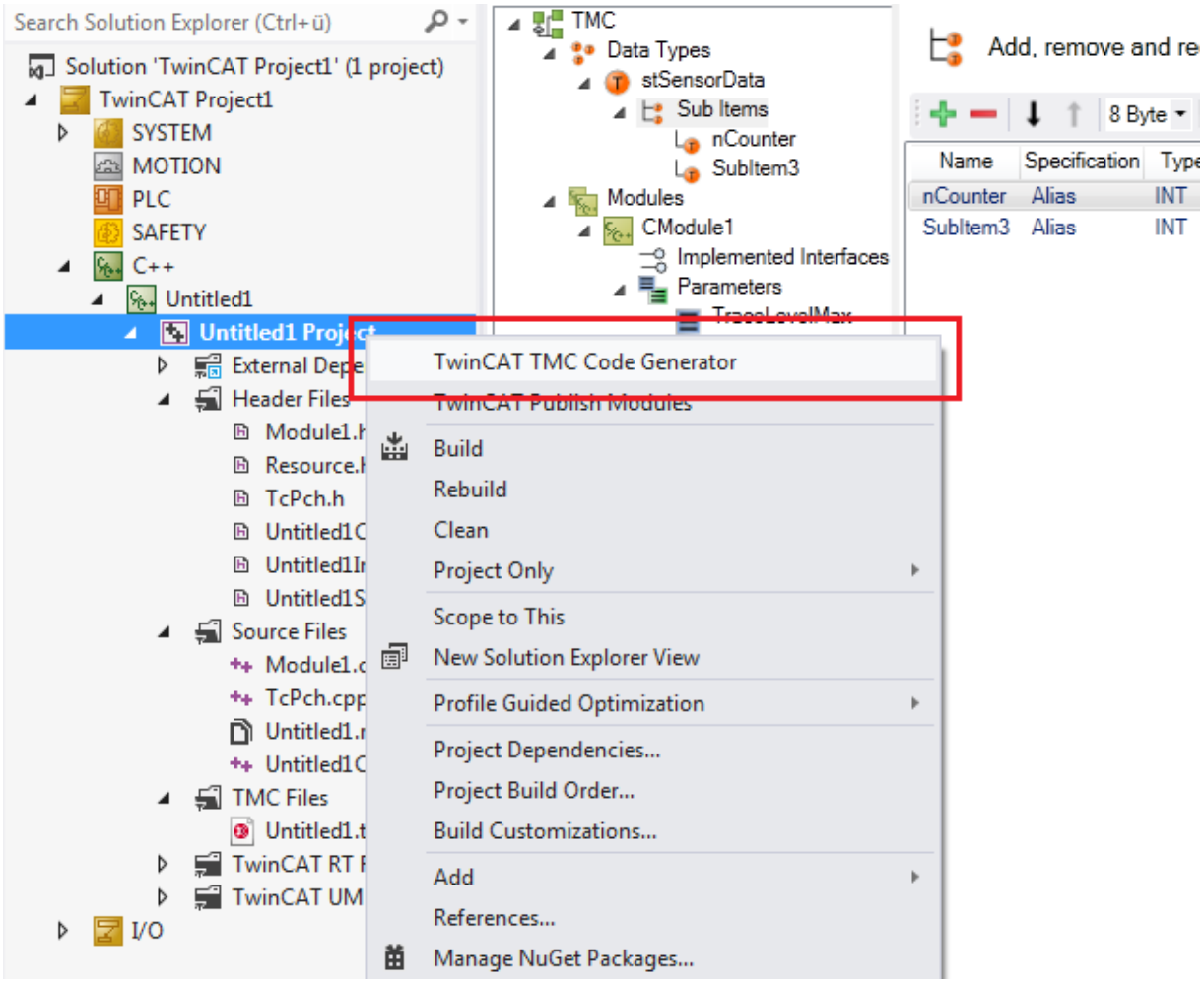
7. Mittels Doppelklick auf das Unterelement können Sie die Eigenschaften bearbeiten. Geben Sie dem Unterelement einen neuen Namen und wählen Sie einen geeigneten Datentyp



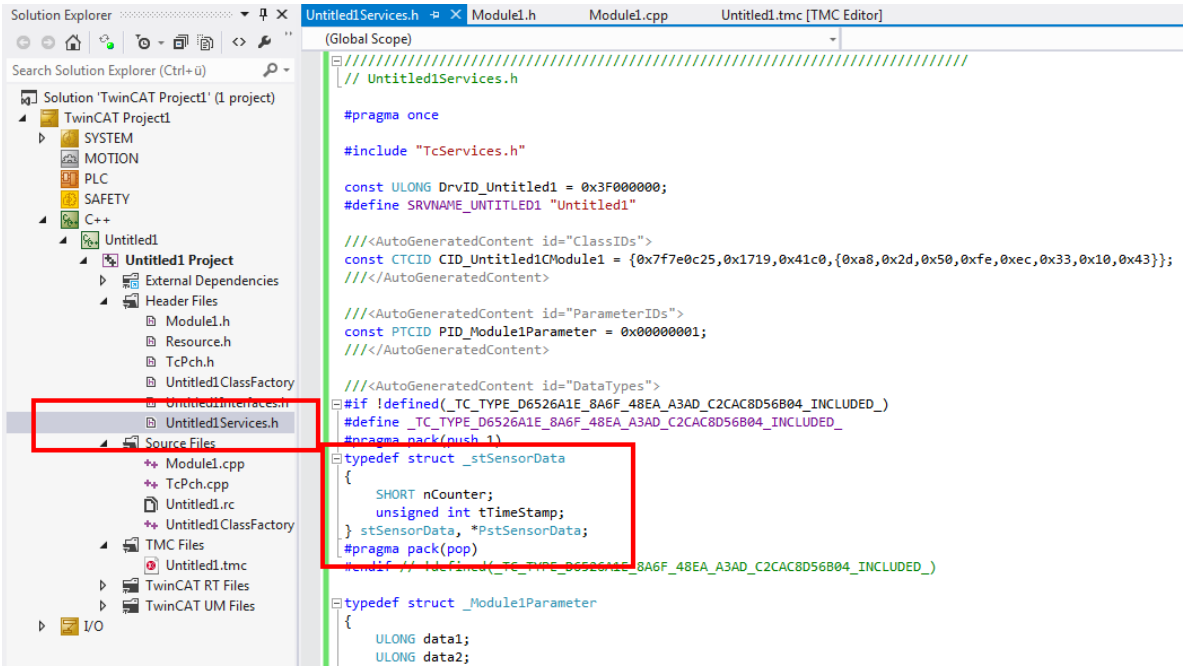
8. Geben Sie den anderen Unterelementen einen neuen Namen und wählen einen geeigneten Datentyp:
9. Speichern Sie Ihre in der TMC-Datei vorgenommenen Änderungen.

Schritt 2: Starten Sie den TwinCAT TMC Code Generator, um einen Code für die Modulbeschreibung zu erzeugen.

10. Klicken Sie mit der rechten Maustaste auf Ihre Projektdatei und wählen Sie **„TwinCAT TMC Code Generator“**, um den Quellcode Ihres Datentyps zu erzeugen:



⇨ Sie werden die Datentypdeklaration in der Modul-Headerdatei „Untitled1Services.h“ sehen



⇨ Wenn Sie einen weiteren Datentyp oder ein weiteres Unterelement hinzufügen, führen Sie den TwinCAT TMC Code Generator erneut aus.

Sehen Sie dazu auch

- Array [▶ 87]
- Enum [▶ 88]
- Struct [▶ 88]
- Schnittstellen [▶ 89]

11.3.3.3 Schnittstellen hinzufügen / bearbeiten / löschen

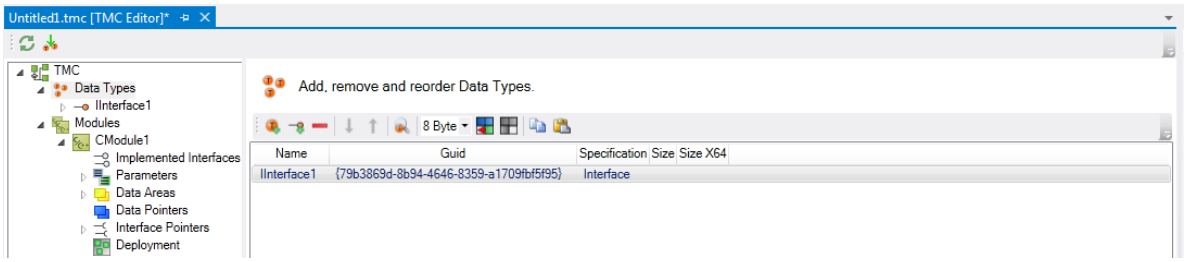
Mit Hilfe des TwinCAT Module Class (TMC) Editors können Schnittstellen eines TwinCAT Moduls hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

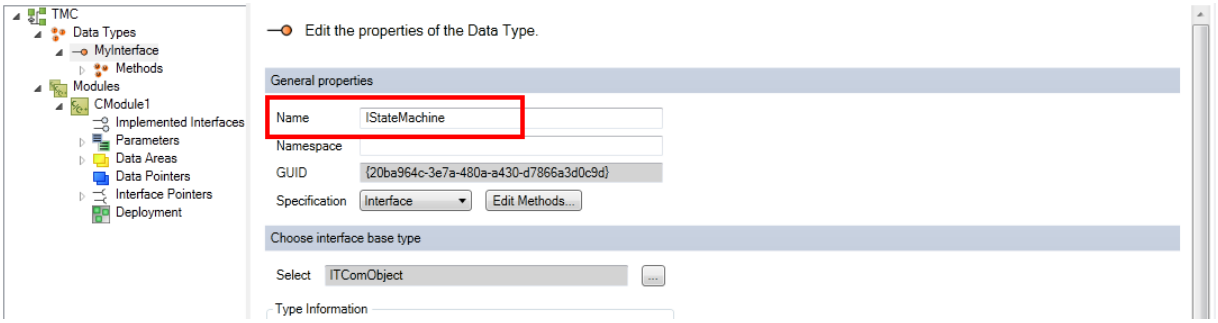
- Schritt 1: Eine neue Schnittstelle [▶ 78] in der TMC-Datei erstellen
- Schritt 2: [▶ 79]Der Schnittstelle in der TMC-Datei Methoden hinzufügen
- Schritt 3: Verwenden Sie die Schnittstelle [▶ 80], indem sie diese zu „Implemented Interfaces“ des Moduls hinzufügen.
- Schritt 4: Starten Sie den TwinCAT TMC [▶ 82] Code Generator, um einen Code für die Modulbeschreibung zu erzeugen.
- Optionale Änderung der Schnittstelle [▶ 82]

Schritt 1: Eine neue Schnittstelle erzeugen

1. Nach dem Starten des TMC Editors den Knoten **„Data Types“** auswählen.
2. Die Liste der Schnittstellen wird mittels Klicken auf „Add a new interface“ um eine neue Schnittstelle erweitert.
   ⇨ Daraufhin wird **„IInterface1“** als neuer Eintrag aufgeführt:



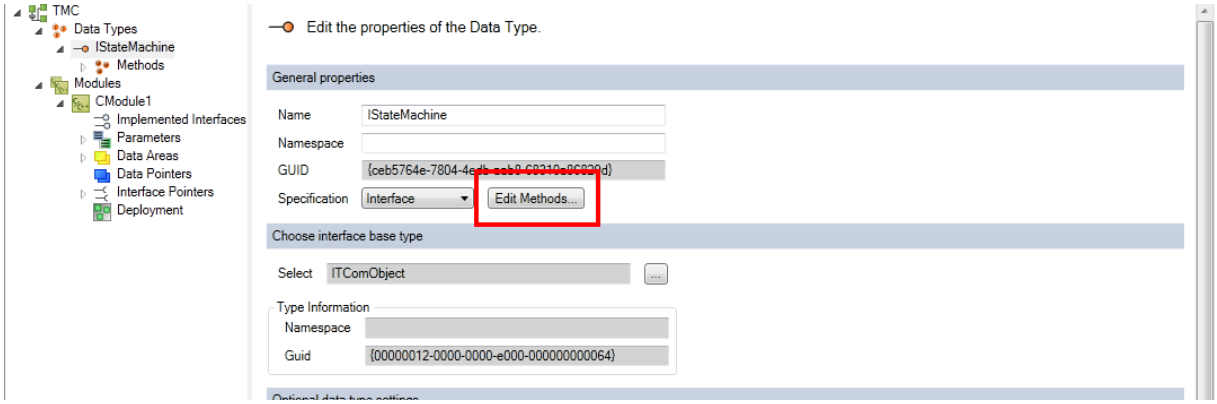
3. Entweder wählen Sie den entsprechenden Knoten im Baum oder machen einen Doppelklick auf die Zeile in der Tabelle, um die Einzelheiten zu öffnen



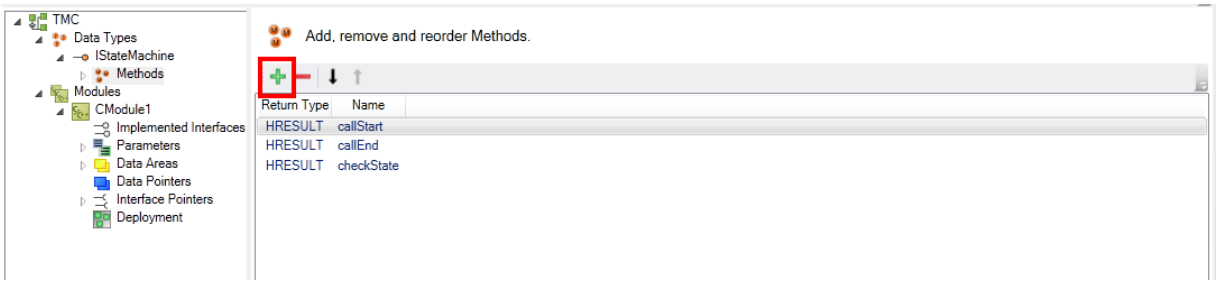
4. Geben Sie einen aussagekräftigeren Namen ein - in diesem Beispiel **„IStateMachine“**.

Schritt 2: Fügen Sie der Schnittstelle Methoden hinzu

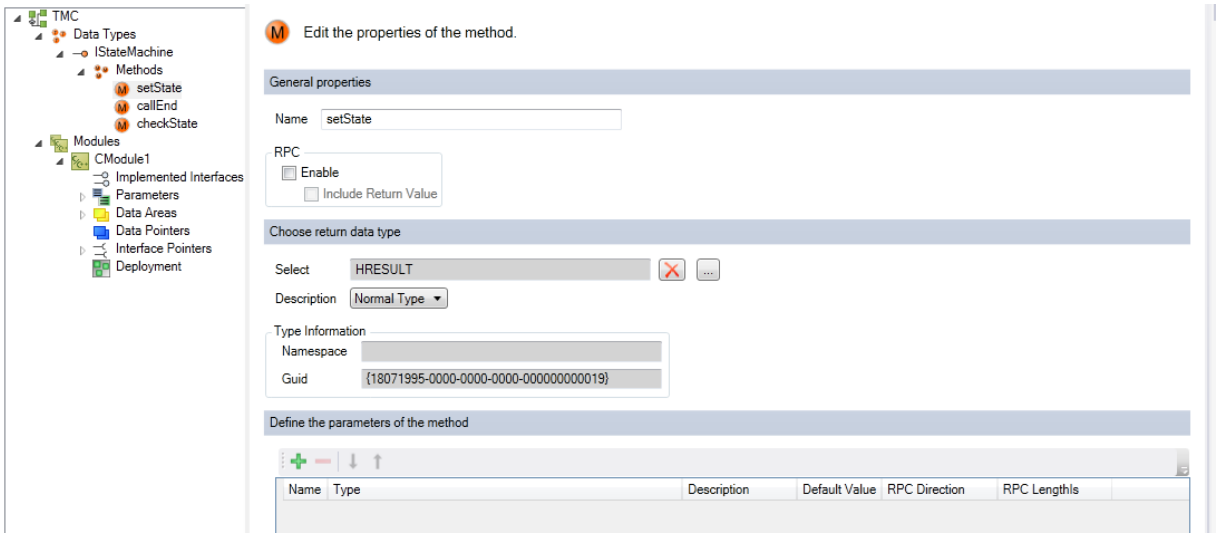
5. Klicken Sie auf **„Edit Methods...“**, um eine Liste der Methoden dieser Schnittstelle zu erhalten:



6. Klicken Sie auf die „+“ Schaltfläche um eine neue standardmäßige Methode **„Method1“** zu erzeugen:



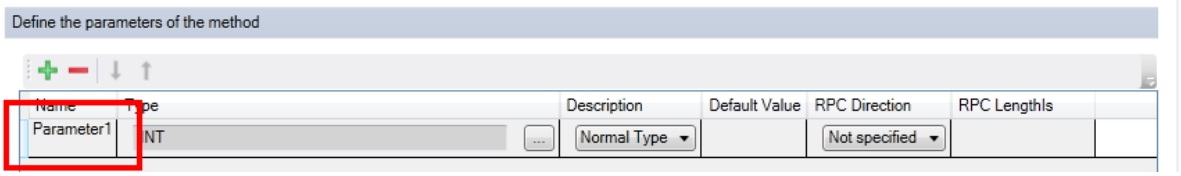
7. Doppelklicken Sie auf die Methode oder wählen den Knoten im Baum aus, um Einzelheiten zu öffnen
8. Geben Sie der standardmäßigen „Method1“ einen aussagekräftigeren Namen.
9. Anschließend können Sie mit einem Klick auf „Add a new parameter“ Parameter hinzufügen / Parameter der Methode „SetState“ bearbeiten



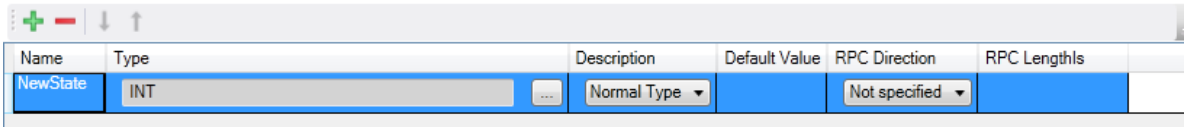
⇨ Standardmäßig wird der neue Parameter „Parameter1“ als „Normal Type“ „INTEGER“ erzeugt.

10. Durch einen Klick auf den Namen „Parameter1“ kann dieser bearbeitet werden.

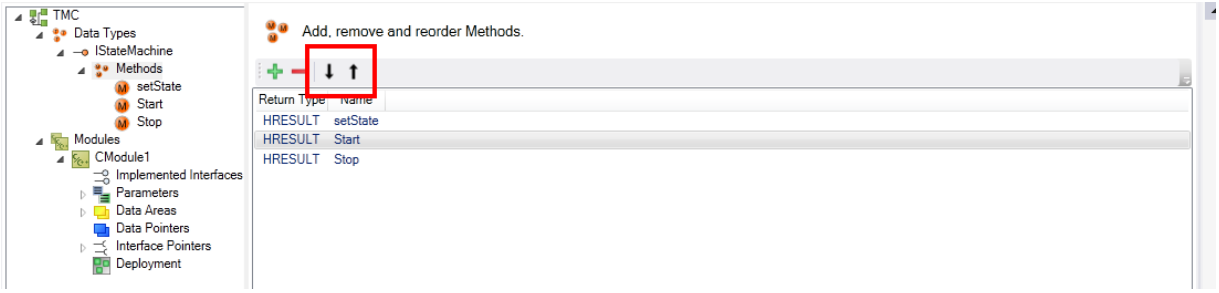
⇨ Der „Normal Type“ kann auch in „Pointer“ geändert werden usw. - auch kann der Datentyp selber ausgewählt werden



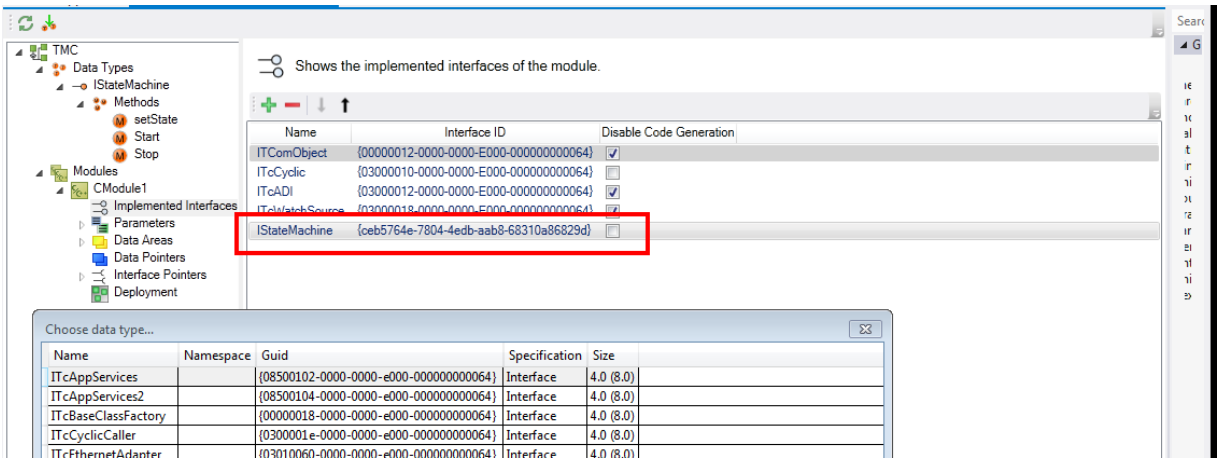
⇨ In diesem Falle ist „NewState“ der neue Name - die übrigen Einstellungen werden nicht geändert



11. Durch Wiederholen des Schritts 2 „Methoden zur Schnittstelle hinzufügen“ werden alle Methoden aufgelistet - mit Hilfe der „nach oben“ / „nach unten“ Schaltfläche können die Methoden neu geordnet werden.

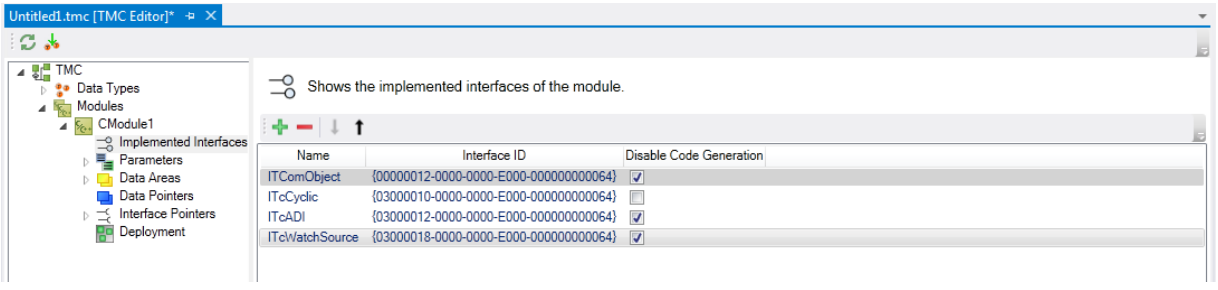


12. Die Schnittstelle ist bereit durch Ihr Modul implementiert zu werden.

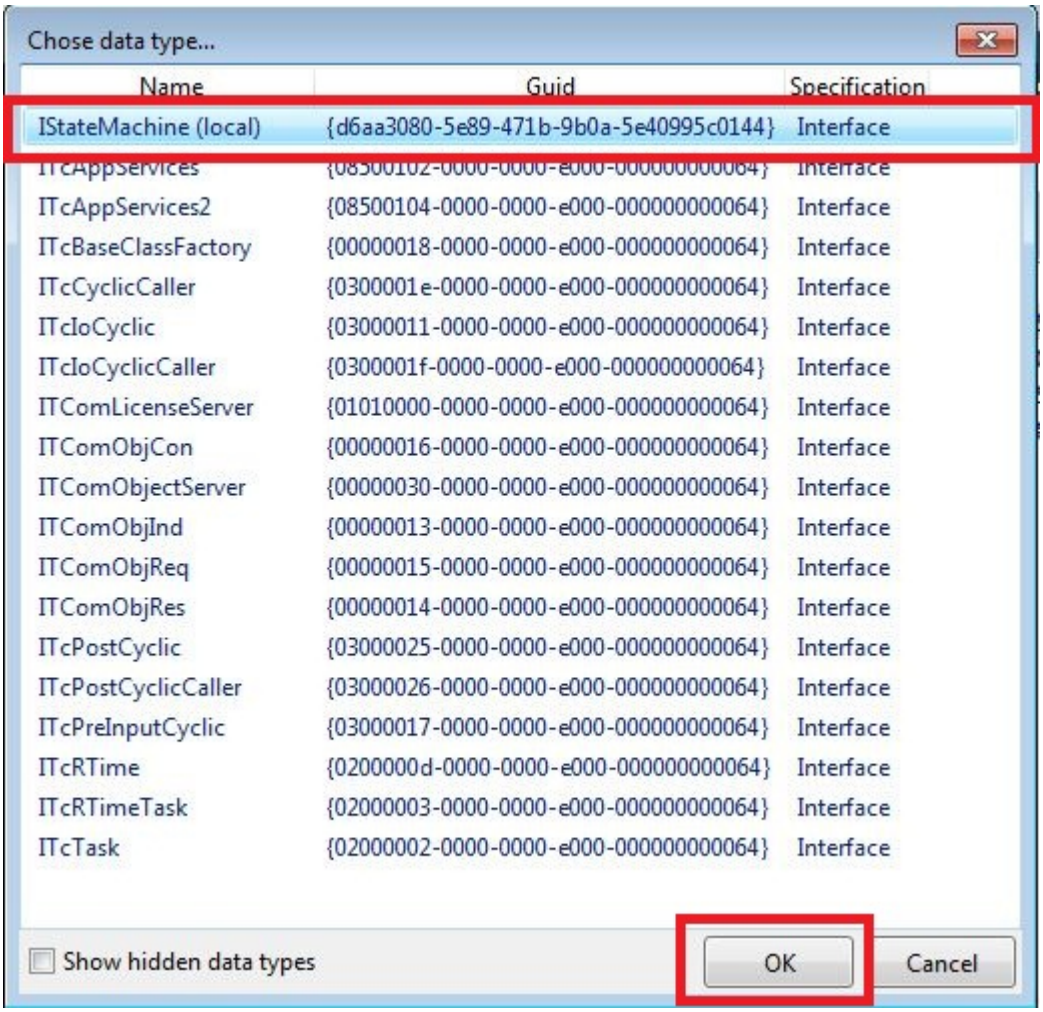


Schritt 3: Die neue Schnittstelle zu „Implemented Interfaces“ hinzufügen

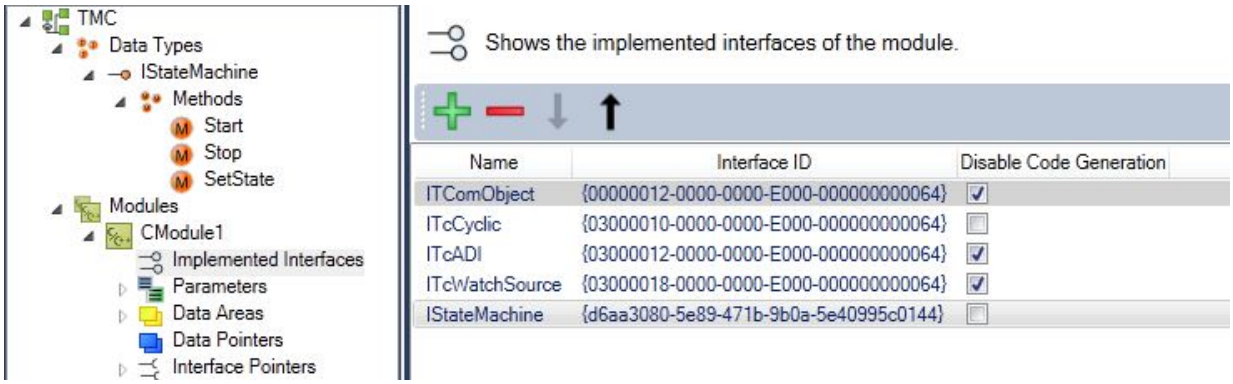
13. Wählen Sie das Modul, das mit der neuen Schnittstelle erweitert werden soll - in diesem Falle wählen Sie das Ziel **„Modules->CModule1“**.
14. Die Liste der implementierten Schnittstellen wird mittels Klicken auf die „+“ Schaltfläche mit „Add a new interface to the module“ um eine neue Schnittstelle erweitert.



15. Alle verfügbaren Schnittstellen werden aufgeführt - wählen Sie die neue Vorlage **„IStateMachine“** und beenden Sie mit „OK“

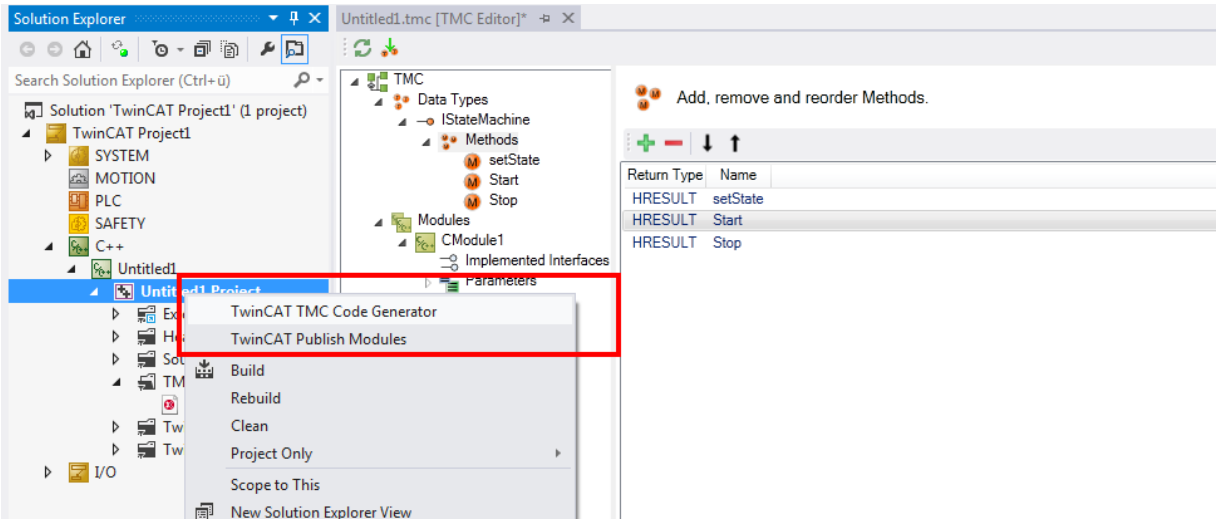


⇨ In Folge dessen ist nun die neue Schnittstelle „IStateMachine“ Teil der Modulbeschreibung.

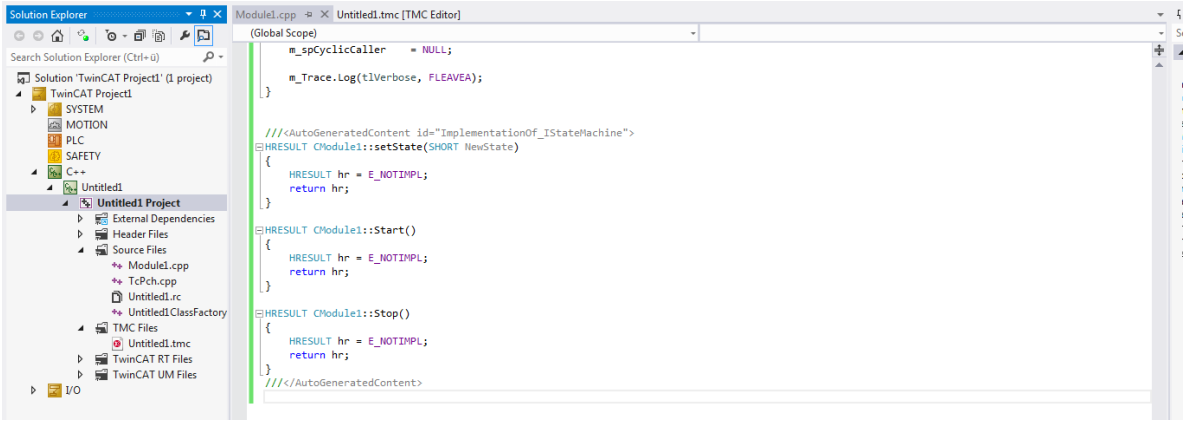


Schritt 4: Starten Sie den TwinCAT TMC Code Generator, um einen Code für die Modulbeschreibung zu erzeugen.

1. Um den C/C++ Code anhand von dieser Modulbeschreibung zu generieren, klicken Sie mit der rechten Maustaste in das C/C++ Projekt und wählen dann den **„TwinCAT TMC Code Generator“**



⇨ Daraufhin enthält das Modul „Module1“ die neuen Schnittstellen
CModule1: Start()
CModule1: Stop()
CModule1: SetState(SHORT NewState)



2. Fertig - der benutzerdefinierte Code kann nun in diesen Bereich eingefügt werden.

Optionale Änderung der Schnittstelle

HINWEIS! Im Falle von Änderungen an der Schnittstelle (z.B. die Parameter einer Methode werden später erweitert) wird benutzerdefinierter Code nie gelöscht. Stattdessen wird die bestehende Methode lediglich mit einem Kommentar versehen, wenn der TMC Code Generator die Methoden nicht mappen kann.

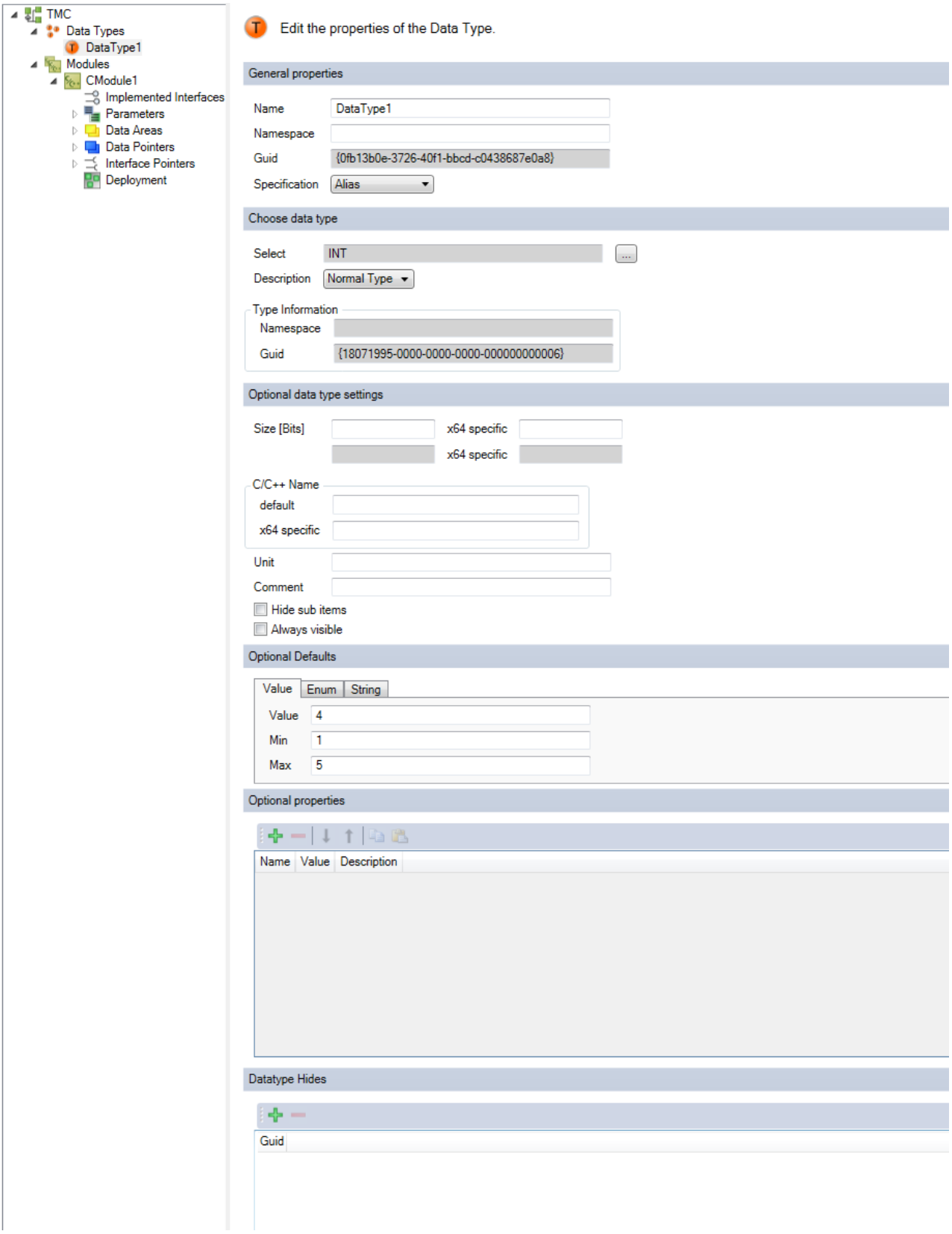
```
///<AutoGeneratedContent id="ImplementationOf_IStateMachine">
HRESULT CModule1::SetState(SHORT SetState, bool bRun)
{
    HRESULT hr = E_NOTIMPL;
    return hr;
}
///</AutoGeneratedContent>

///<AutoGeneratedContent id="Obsolete_ImplementationOf_IStateMachine">
//HRESULT CModule1::SetState(SHORT SetState)
//{
//  HRESULT hr = E_NOTIMPL;
//
//  //custom code
//  nState = SetState;
//
//  return hr;
//}
//
///</AutoGeneratedContent>
```

Sehen Sie dazu auch

- Implementierte Schnittstellen [▶ 93]

11.3.3.4 Datentypeigenschaften

Die Eigenschaften von Datentypen bearbeiten



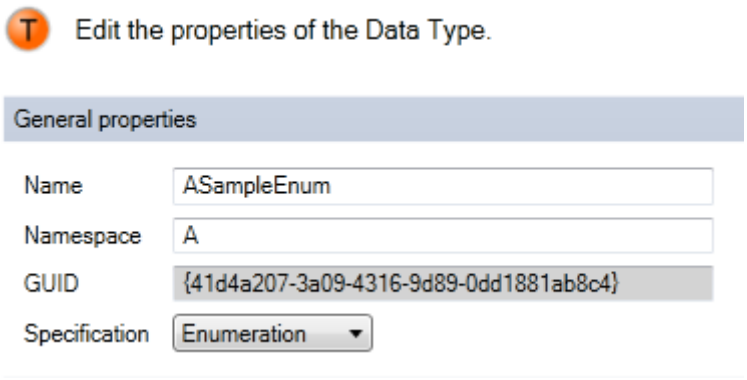
Allgemeine Eigenschaften

Name: Benutzerdefinierte Name des Datentyps

Hinweis	**Namenskonflikt** Bitte verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen. Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namenskollisionen kommen.

Namespace: Benutzerdefinierte Namensraum des Datentyps

Beachten Sie, dass dieser **nicht** einem C Namensraum zugeordnet wird. Er wird als Präfix Ihres Datentyps verwendet werden. Nehmen wir eine Aufzählung mit einem Namensraum „A“ als Beispiel



dann wird der folgende Code generiert:

```
///<AutoGeneratedContent id="DataTypes">
#if !defined(_TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_)
#define _TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_
enum A_ASampleEnum : SHORT {
    One,
    Two,
    Three
};
#endif // !defined(_TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_)
```

Möglicherweise möchten Sie den Namensraumnamen dem Aufzählungselement manuell als Präfix hinzufügen:

```
#if !defined(_TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_)
#define _TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_
enum B_BSampleEnum : SHORT {
    B_one,
    B_two,
    B_three
};
#endif // !defined(_TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_)
```

GUID: Eindeutige ID des Datentyps

Specification: Festlegung des Datentyps

- **Alias**: Ein Alias eines Standarddatentyps (z.B. INT) erzeugen
- Array [▶ 87]: Ein benutzerdefiniertes Array erstellen
- Enumeration [▶ 88]: Eine benutzerdefinierte Aufzählung erstellen
- Struct [▶ 88]: Eine benutzerdefinierte Struktur erzeugen
- Interface [▶ 89]: Eine neue Schnittstelle erzeugen

Datentyp auswählen

Select: Datentyp auswählen - hierbei kann es sich um Basisdatentypen von TwinCAT oder um benutzerdefinierte Datentypen handeln.
Es sind Datentypen äquivalent zu den SPS-Datentypen definiert (wie TIME, LTIME usw.). Siehe Datentypen der SPS für weitere Auskünfte.

Description: Den Typ als Zeiger, Referenz oder Wert mittels entsprechender Auswahl definieren

- Normaler Typ
- Zeiger
- Zeiger auf Zeiger
- Zeiger auf Zeiger auf Zeiger
- eine Referenz

Typinformation

- **Namespace:** Für ausgewählten Datentyp definiert
- **GUID**: Eindeutige ID des ausgewählten Datentyps

Optionale Datentypeinstellungen

Size [Bits]: Größe in Bits (weiße Felder) und in „Byte.Bit“ Notation (graue Felder). Für x64-Plattform kann eine andere Größe festgelegt werden.

C/C++ Name: Im generierten C++ Code verwendeter Name. Der TMC Code Generator wird die Deklaration nicht generieren, sodass benutzerdefinierter Code für diesen Datentyp bereitgestellt werden kann. Darüber hinaus kann für x64 ein anderer Name festgelegt werden.

Unit: Eine Einheit der Variablen

Comment: Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist

Hide sub items: Wenn der Datentyp über Unterelemente verfügt, dann wird der Systemmanager keinen Zugriff auf die Unterelemente gewähren. Dies sollte z.B. im Falle großer Arrays verwendet werden.

Always visible: Datentyp wird im Typensystem importiert, selbst dann wenn er von keiner Modulklasse referenziert wird.

Optionale Standardeinstellungen

Die Standardeinstellungen können in Funktion des Datentyps definiert werden.

Optionale Eigenschaften

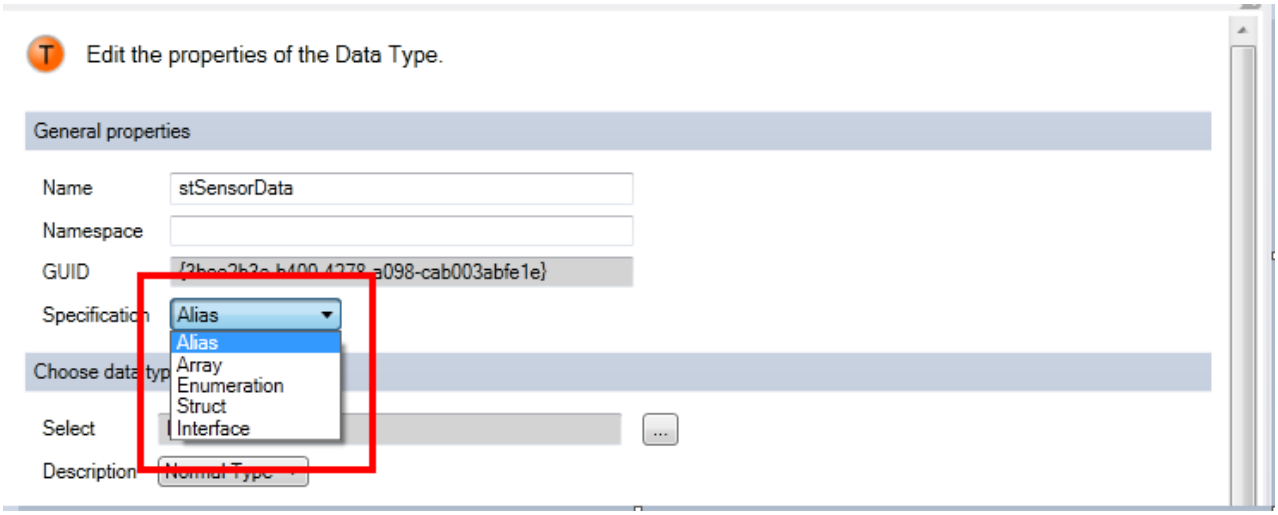
Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Datentyps.
Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

Datentypausblendungen

Aufgelistete GUIDs verweisen auf Datentypen, die von diesem Datentypen ausgeblendet werden.
Normalerweise werden die GUIDs vorheriger Versionen dieses Datentyps hier bei jeder Änderung automatisch eingefügt.

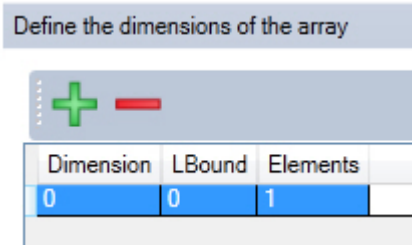
11.3.3.5 Spezifikation

In diesem Abschnitt wird die Spezifikation von Datentypen beschrieben.



11.3.3.5.1 Array

Array: Ein benutzerdefiniertes Array erstellen



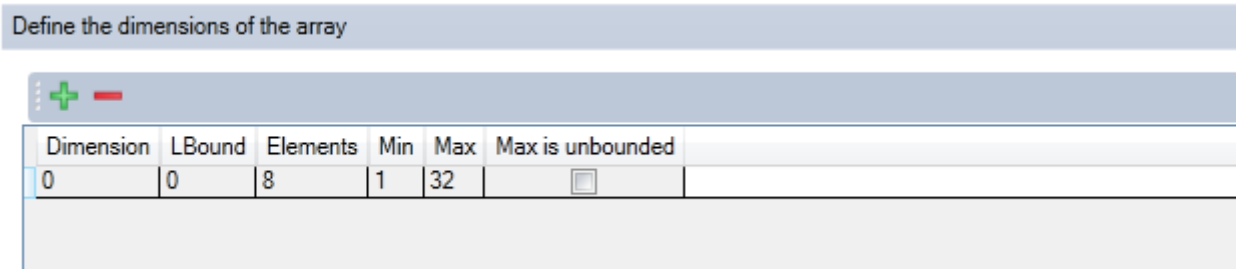
Es wird ein neuer Dialog eingeblendet, um Array-Elemente hinzuzufügen (+) oder zu entfernen (-).

Dimension: Dimension des Array

LBound: Linke Grenze des Array (Standardwert = 0)

Elemente: Menge der Elemente

Dynamische Arrays für Parameter und Datenzeiger



Im Falle von Parameter [▶ 93] und Datenzeigern [▶ 108] unterstützt TwinCAT 3 Arrays mit dynamischer Größe.

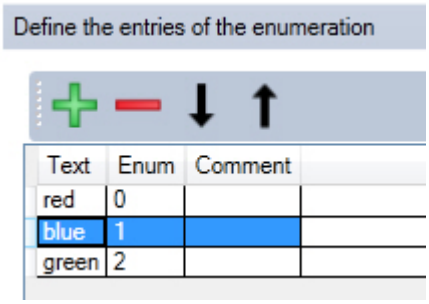
Min: Mindestgröße des Arrays

Max: Maximale Größe des Arrays

Max is unbounded: Zeigt an, dass es keine obere Grenze für die Array-Größe gibt

11.3.3.5.2 Enum

Enumeration: Eine benutzerdefinierte Aufzählung erstellen



Es wird ein neuer Dialog eingeblendet, um ein Element hinzuzufügen (+) oder zu entfernen (-). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.

Hinweis	**Eindeutige Namen für Aufzählungselemente erforderlich** Beachten Sie, dass die Aufzählungselemente eindeutig benannt sein müssen, da ansonsten der generierte C++ Code ungültig ist.

Text: Aufzählungselement

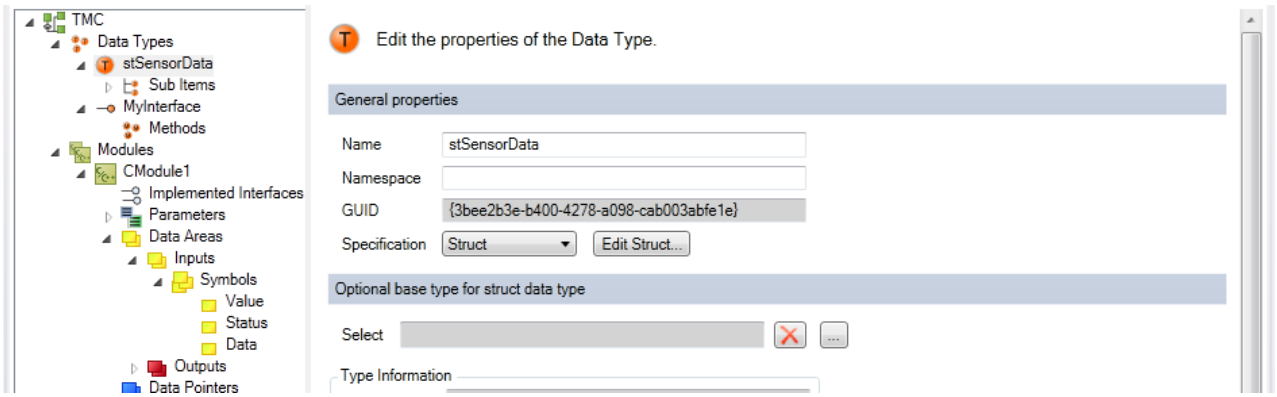
Enum: Geeigneter Integerwert

Comment: Optionaler Kommentar

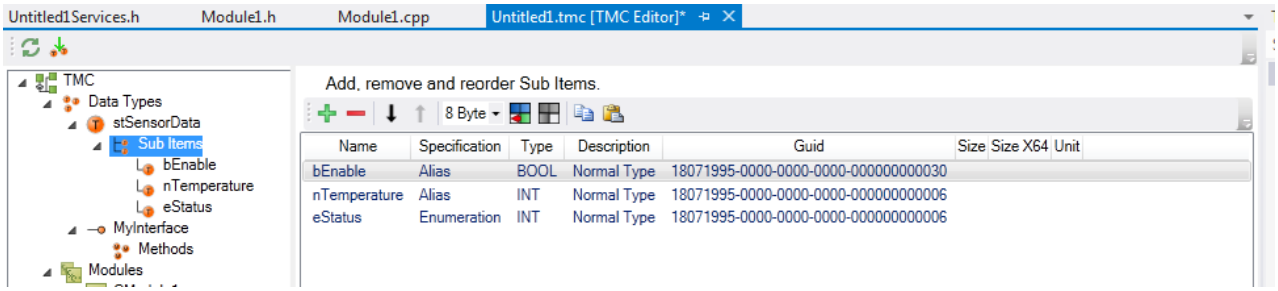
11.3.3.5.3 Struct

Struct: Eine benutzerdefinierte Struktur erstellen

Wählen Sie bitte den „Sub Items“-Knoten oder klicken Sie auf die „Edit Struct“-Schaltfläche, um zu dieser Tabelle zu wechseln:



Es wird ein neuer Dialog eingeblendet, um ein Element hinzuzufügen (+) oder zu entfernen (-). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.



Name: Name des Elements

Specification: Ein Struct kann Alias, Arrays oder Aufzählungen enthalten

Type: Typ der Variablen

Size: Größe und Offset des Unterelements.

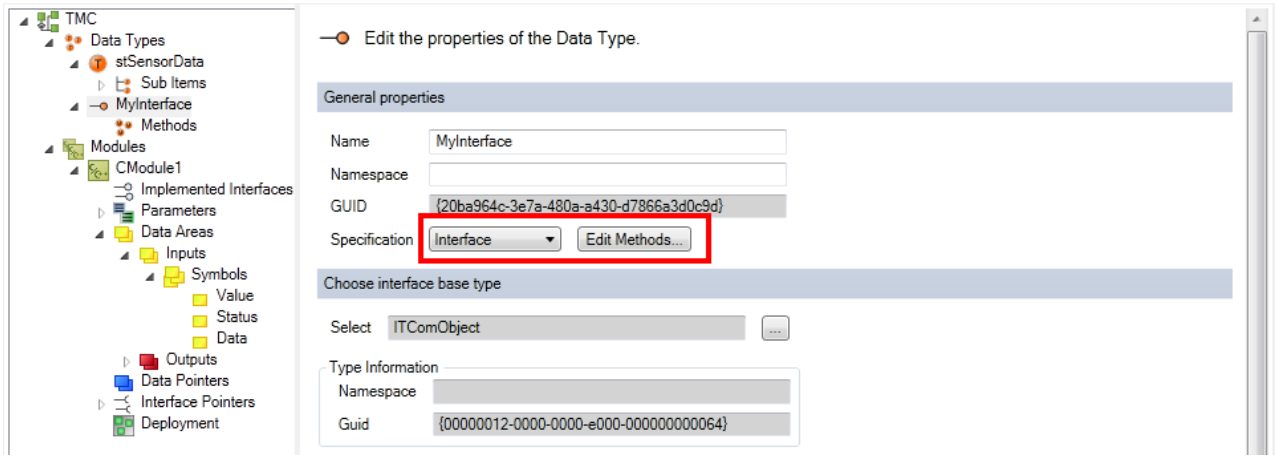
Size X64: Andere Größe für x64 Plattform wird zusätzlich bereitgestellt.

Unit: Optionale Einheit

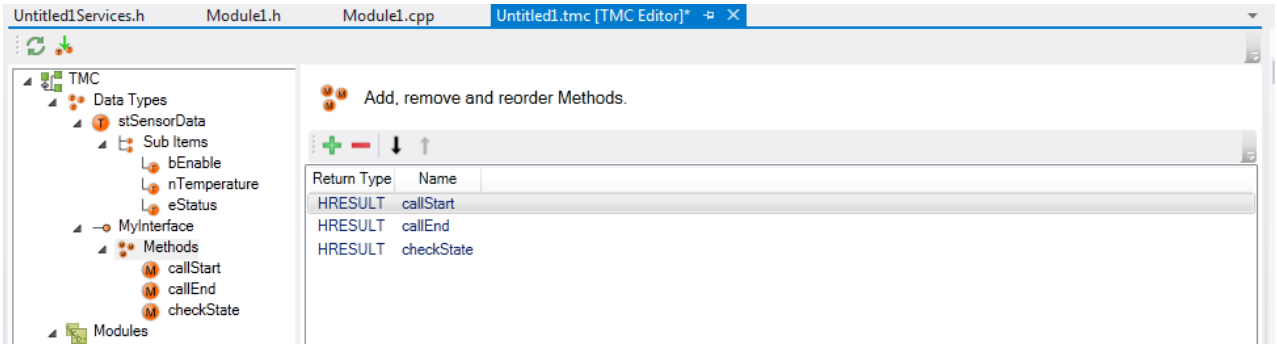
Mittels Auswahl des Datentyps oder Doppelklick auf den Tabelleneintrag werden die Details der Konfigurationsseite des Unterelements eingeblendet. Ähnlich wie Datentypeigenschaften [▶ 84].

11.3.3.5.4 Schnittstellen

Interfaces: Eine benutzerdefinierte Schnittstelle erstellen.

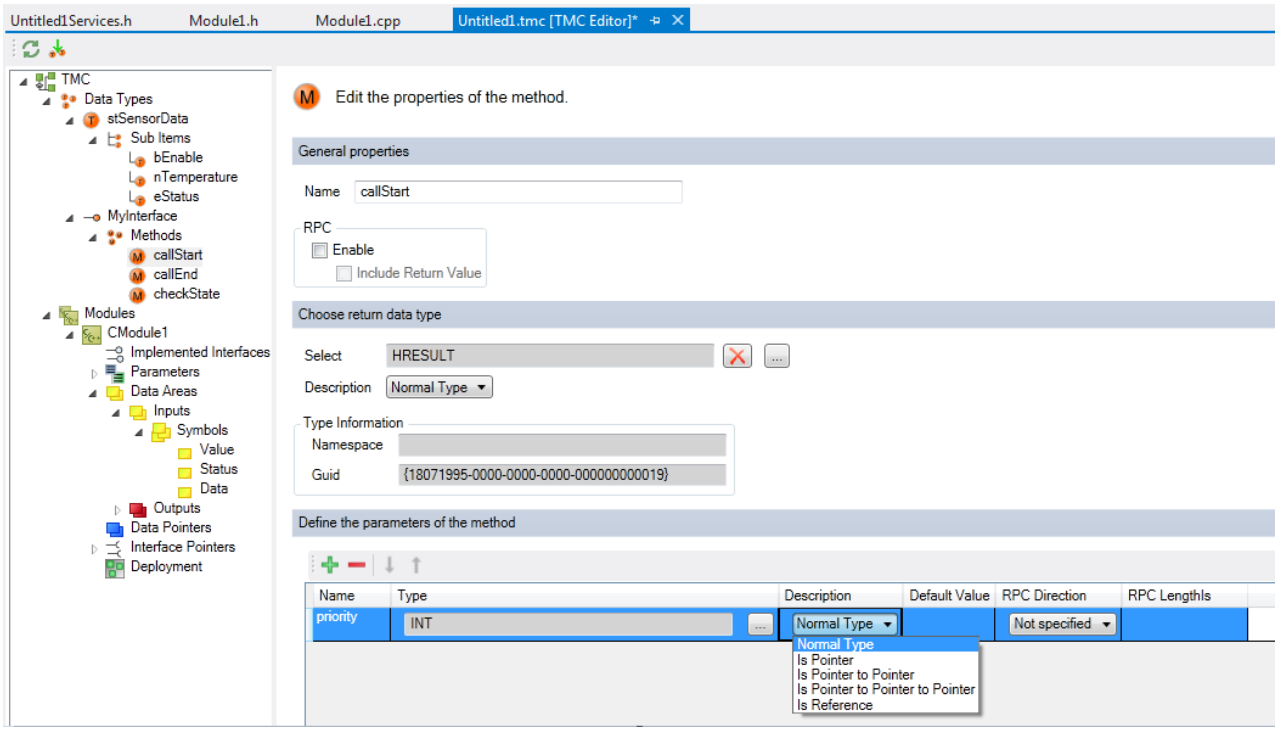


Wählen Sie bitte den „Methods“-Knoten oder klicken Sie auf die „Edit Methods“-Schaltfläche, um zu dieser Tabelle zu wechseln:



Parameter der Methode

Wählen Sie den Knoten der Methode oder doppelklicken Sie auf den Eintrag, um die Details der Methode einzusehen.



Name: Der Name der Methode

RPC enable: Freigabe von „Remoteprozeduraufrufen“ von außerhalb dieser Methode.

- **Include Return Value**: Freigabe der Weitergabe des Rückgabewerts der Methode

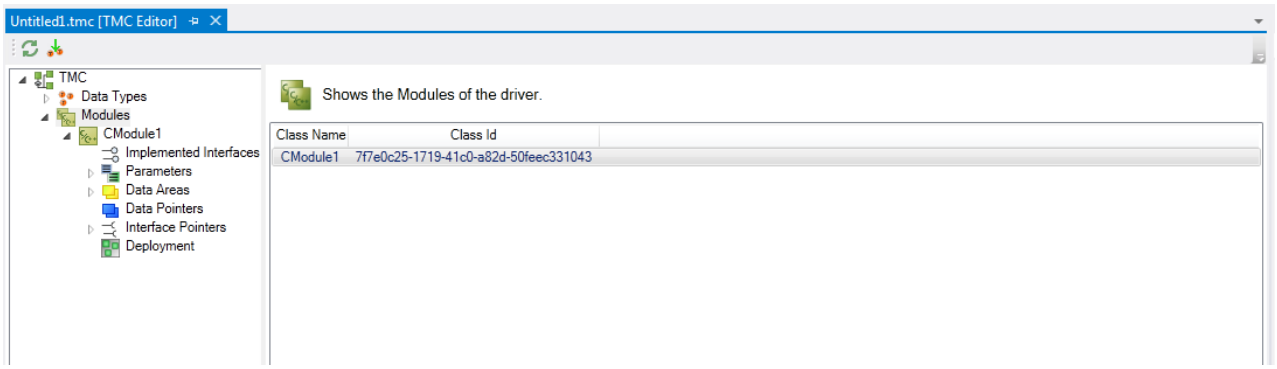
Felder entsprechen denjenigen der Datentypeigenschaften [▶ 84].

Parameter der Methode definieren

- Name
- Type: Bekannt aus den Datentypeigenschaften [▶ 84]
- Description: Bekannt aus den Datentypeigenschaften [▶ 84]
- Default Value: Standardwert dieses Parameters; es sind nur Zahlen erlaubt.
- RPC-Direction: Wie im Falle von SPS-Funktionsblöcken kann jeder Parameter entweder IN, OUT oder INOUT sein. Darüber hinaus kann er als NONE definiert werden, damit dieser Parameter bei Remoteprozeduraufrufen (RPC) ignoriert wird.

11.3.4 Module

Modules: Zeigt die Module des Treibers

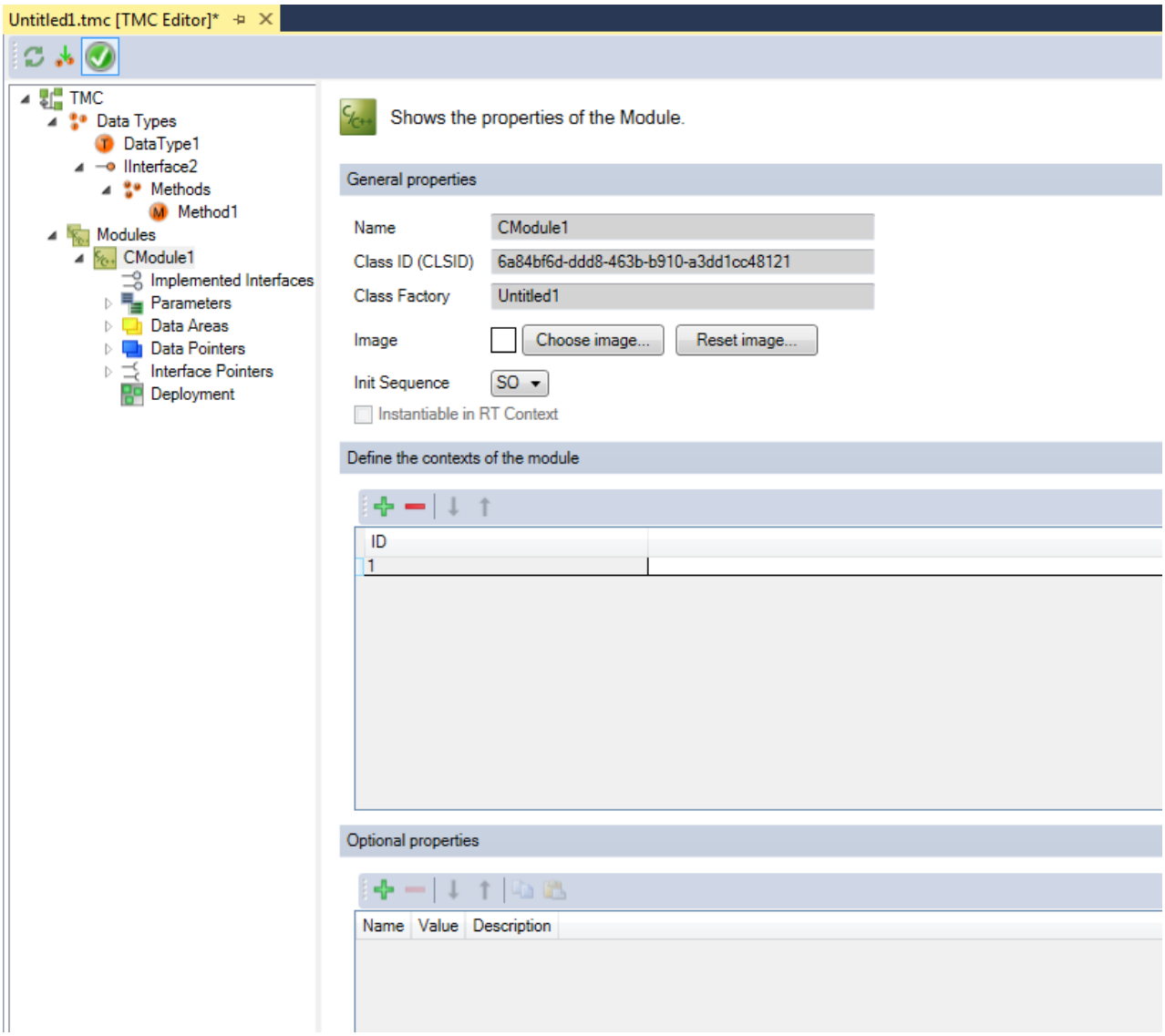


Class Name: Name des Moduls

Class ID: Eindeutige ID des Moduls

Module Properties:

Ein Klick auf den Knoten im Baum oder der Zeile in der Tabelle öffnet die Moduleigenschaften.



Allgemeine Eigenschaften

Name: Name des Moduls

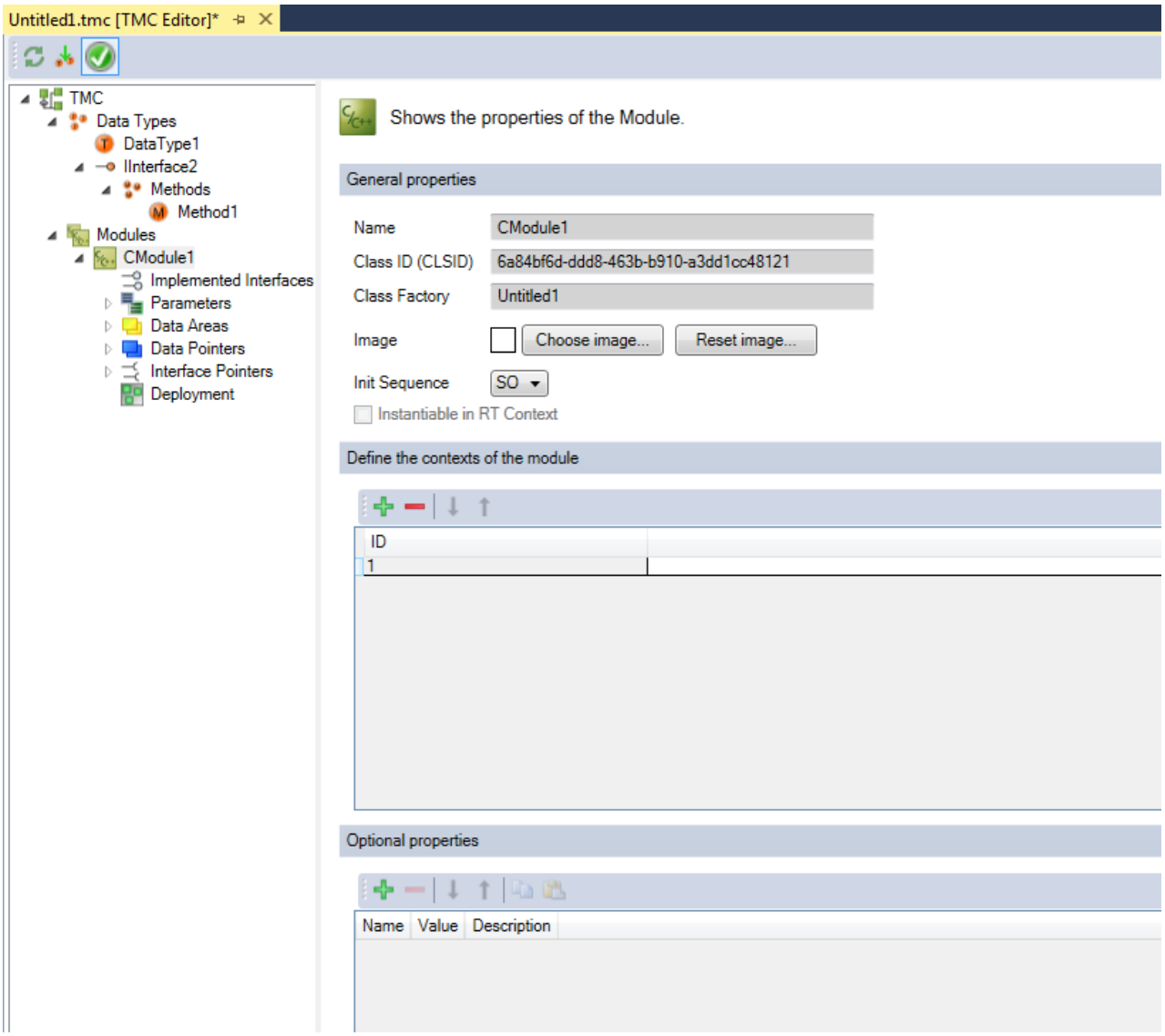
Class ID: Eindeutige ID des Moduls

Choose Image: Ein 16x16 Pixel-Bitmap-Symbol einfügen

Reset image: Modulbild auf Standardwert zurücksetzen

Init sequence: Start der Zustandsmaschine. (Siehe Objekt [▶ 113] des Instance Configurators für weitere Auskünfte.)

Instantiable in RT Context: Zeigt an, ob dieses Modul unter Echtzeitkontext instanziiert werden kann, siehe TwinCAT Module Klassenassistent [▶ 66]



Die Kontexte des Moduls festlegen

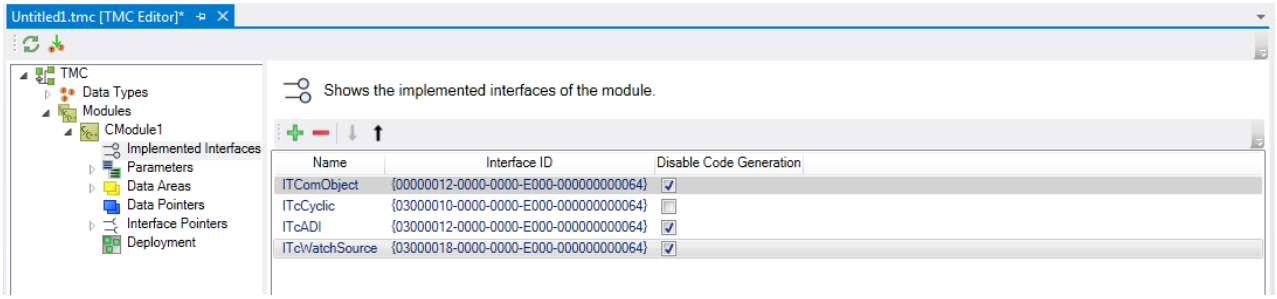
Sie können für das Modul Kontexte hinzufügen (+) oder entfernen (-). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.
Die Kontext-Id muss ein Integer verschieden von 0 sein.

Optionale Eigenschaften

Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Moduls.
Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

11.3.4.1 Implementierte Schnittstellen

Implemented Interfaces: Die implementierten Schnittstellen des Moduls ansehen und bearbeiten

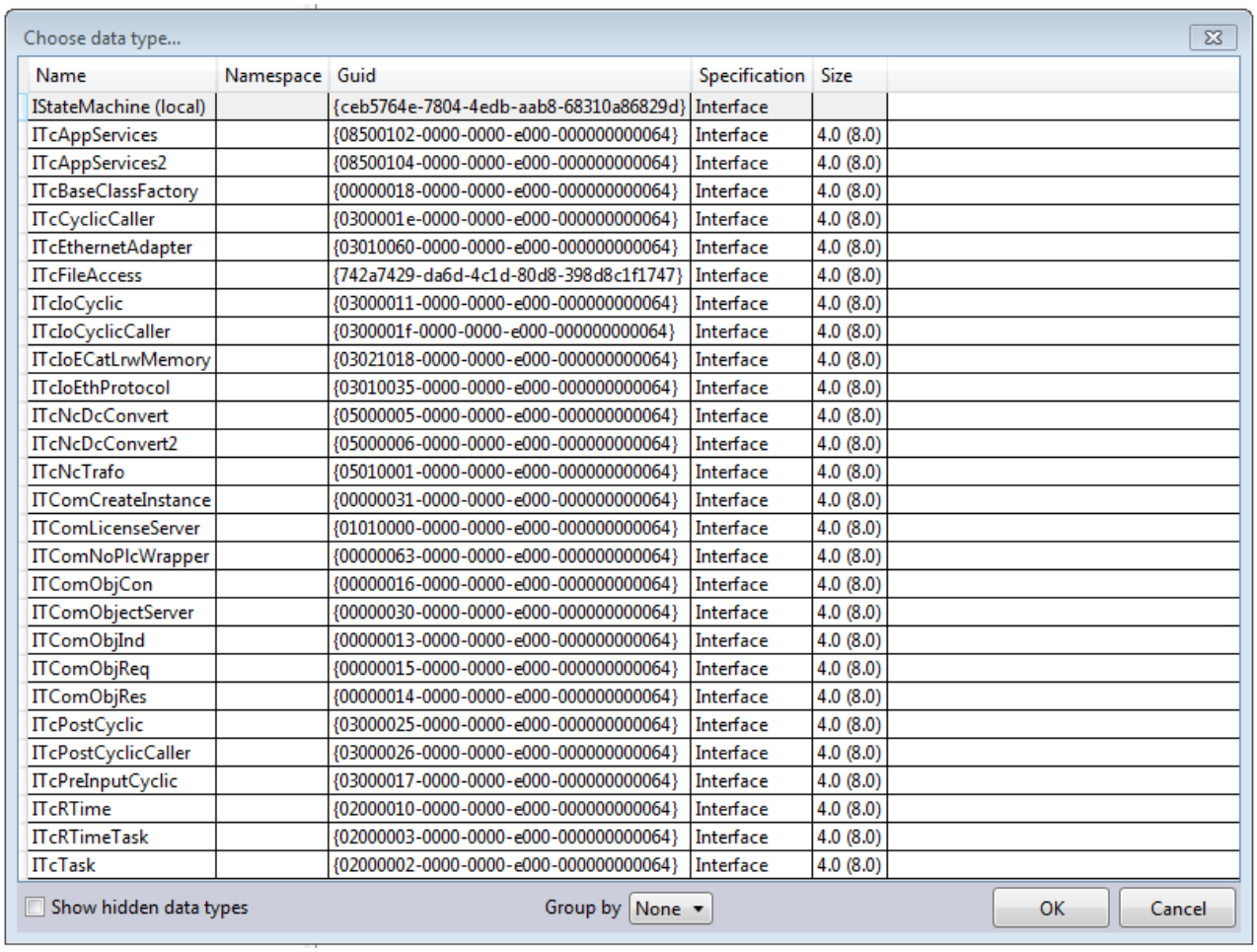


Name: Name der Schnittstelle

Interface ID: Eindeutige ID der Schnittstelle

Disable Code Generation: Die Code-Generierung freigeben/sperren

Sie können für das Modul Kontexte hinzufügen (+) oder entfernen (-). Bearbeiten Sie mit Hilfe der Pfeile die Reihenfolge.



11.3.4.2 Parameter

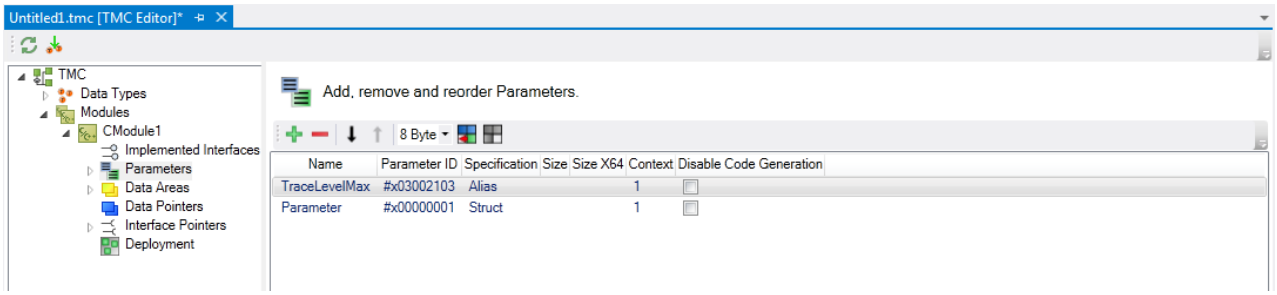
Eine TcCOM-Modulinstanz wird durch verschiedene Parameter konfiguriert.

TwinCAT unterstützt drei Arten von „Parameter IDs“ (PTCID) im Abschnitt „Configure the parameter ID“.

- „user defined“ (Standardwert neuer Parameter): Es wird eine eindeutige Parameter-ID generiert, die im Nutzercode oder in der Instanzkonfiguration zur Festlegung des Parameters verwendet werden kann.
- „Predefined...“: Spezielle PTCIDs bereitgestellt vom TwinCAT 3 System (z.B. TcTraceLevel).
- „Context based…“: Werte des konfigurierten Kontext [▶ 114] diesem Parameter automatisch zuweisen. Die ausgewählte Eigenschaft wird für die PTCPID übernommen. Das überschreibt den definierten Standardparameter und den Instanzkonfigurationsparameter („Parameter (Init)“).

Es folgt eine ausführliche Beschreibung der Parameter und ihrer Konfiguration.

Parameters: Zeigt die implementierten Parameter des Moduls.



Symbol	Funktion
	Einen neuen Parameter hinzufügen
	Löscht den ausgewählten Typ
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben
	Byte Alignment auswählen
	Ausgewählten Datentyp ausrichten
	Datenformat des ausgewählten Datentyps zurücksetzen

Name: Name der Schnittstelle

Parameter ID: Eindeutige ID des Parameters

Specification: Datentyp des Parameters

Size: Größe des Parameters. Für x64 sind andere Größen möglich.

Context: Kontext-ID des Parameters

Disable Code Generation: Die Code-Generierung freigeben/sperren

11.3.4.2.1 Parameter hinzufügen / bearbeiten / löschen

Mit Hilfe des TwinCAT Module Class (TMC) Editors können Eigenschaften und Funktionalitäten einer TwinCAT-Klasse hinzugefügt, bearbeitet und gelöscht werden.

Dieser Artikel beschreibt:

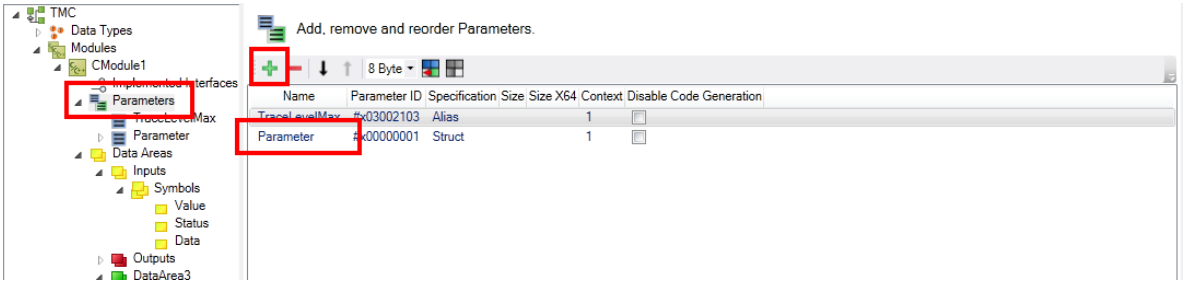
- Schritt 1: Einen neuen Parameter [▶ 95]in der TMC-Datei erstellen

- Schritt 2: Starten Sie den TwinCAT TMC Code Generator [▶ 96], um Code für die Modulbeschreibung in der TMC-Datei zu erzeugen
- Schritt 3: Übergänge der Zustandsmaschine [▶ 97]

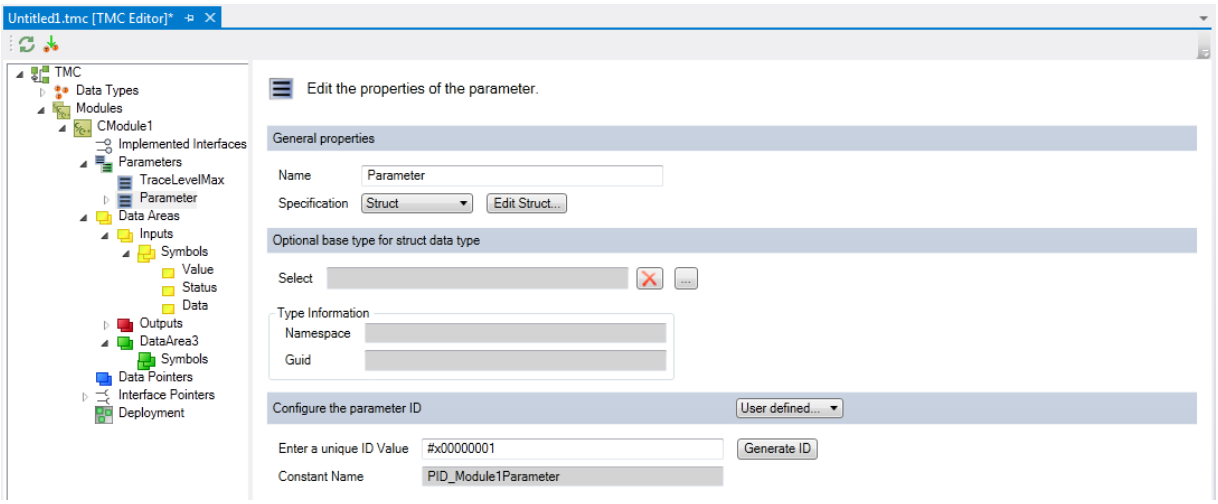
Schritt 1: Neuen Parameter erzeugen

1. Nach dem Starten des TMC Editors das Ziel **„Parameters“** auswählen.
2. Die Liste der Parameter wird mittels Klicken auf die „+“ Schaltfläche „Add a new parameter“ um einen neuen Parameter erweitert.

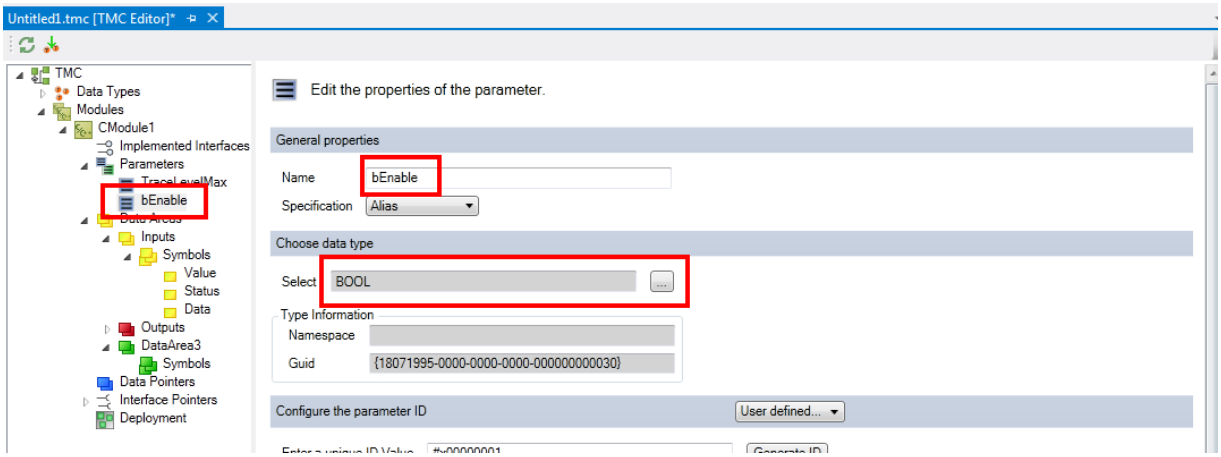
   ⇨ Daraufhin wird ein neuer **„Parameter“** als neuer Eintrag aufgeführt:



3. Wählen Sie „Parameter“ im linken Baum oder doppelklicken Sie auf den rot markierten „Parameter3“ oder wählen Sie den Knoten im Baum, um Details zum neuen Parameter zu erhalten.



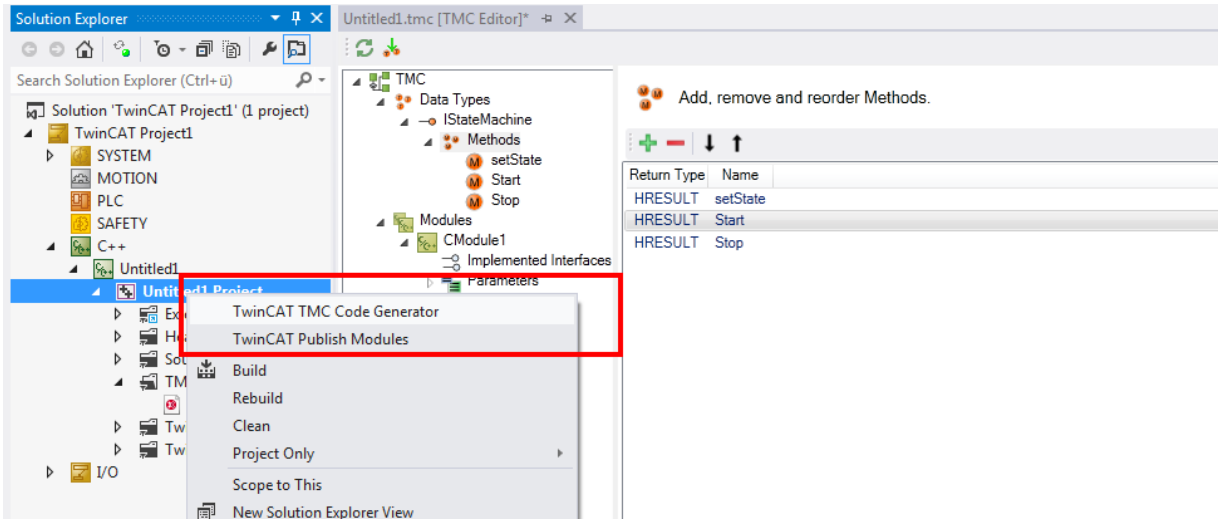
4. Parameter sowie die Datentypen [▶ 72] konfigurieren
5. Geben Sie diesem einen aussagekräftigeren Namen - in diesem Beispiel **„bEnable“** - und wählen den Datentyp „BOOL“.



6. Speichern Sie Ihre in der TMC-Datei vorgenommenen Änderungen.

Schritt 2: Starten Sie den TwinCAT TMC Code Generator um einen Code für die Modulbeschreibung zu erzeugen.

7. Klicken Sie mit der rechten Maustaste auf Ihre Projektdatei und wählen **„TwinCAT TMC Code Generator“**, um den Parameter in Ihrem Quellcode zu erhalten:



⇨ Sie werden die Parameterdeklaration in der Header-Datei „Module1.h“ des Moduls sehen

```
///<AutoGeneratedContent id="Members">
    TcTraceLevel m_TraceLevelMax;
    bool m_bEnable;
    Module1Inputs m_Inputs;
    Module1Outputs m_Outputs;
    Module1DataArea3 m_DataArea3;
    ITcCyclicCallerInfoPtr m_spCyclicCaller;
///</AutoGeneratedContent>
```

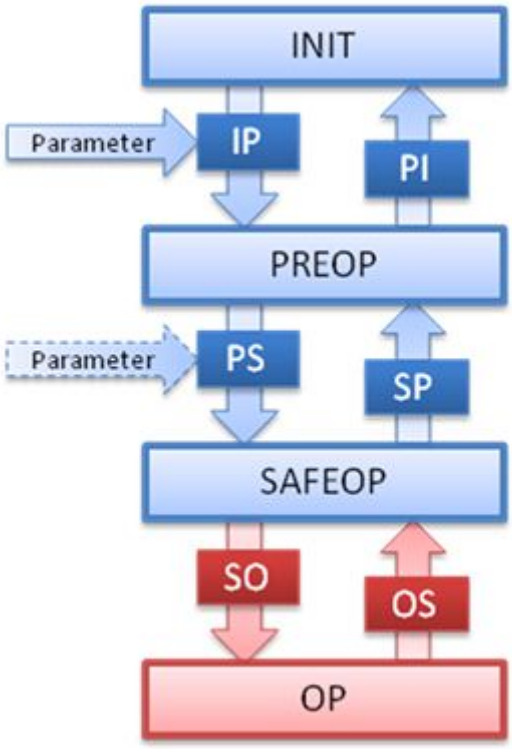
⇨ Die Implementierung des neuen Parameters finden Sie in den get und set Methoden der Modulklasse „module1.cpp“.

```
IMPLEMENT_ITCOMOBJECT(CModule1)
IMPLEMENT_ITCOMOBJECT_SETSTATE_LOCKOP2(CModule1)
IMPLEMENT_ITCADI(CModule1)
IMPLEMENT_ITCWATCHSOURCE(CModule1)


////////////////////////////////////////////////////////////////////////////////
// Set parameters of CModule1
BEGIN_SETOBJPARA_MAP(CModule1)
    SETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="SetObjectParameterMap">
    SETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    SETOBJPARA_VALUE(PID_Module1bEnable, m_bEnable)
    SETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
END_SETOBJPARA_MAP()

////////////////////////////////////////////////////////////////////////////////
// Get parameters of CModule1
BEGIN_GETOBJPARA_MAP(CModule1)
    GETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="GetObjectParameterMap">
    GETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    GETOBJPARA_VALUE(PID_Module1bEnable, m_bEnable)
    GETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
END_GETOBJPARA_MAP()
```

Um einen weiteren Parameter hinzuzufügen, verwenden Sie erneut den TwinCAT TMC Code Generator.

Schritt 3: Übergänge der Zustandsmaschine

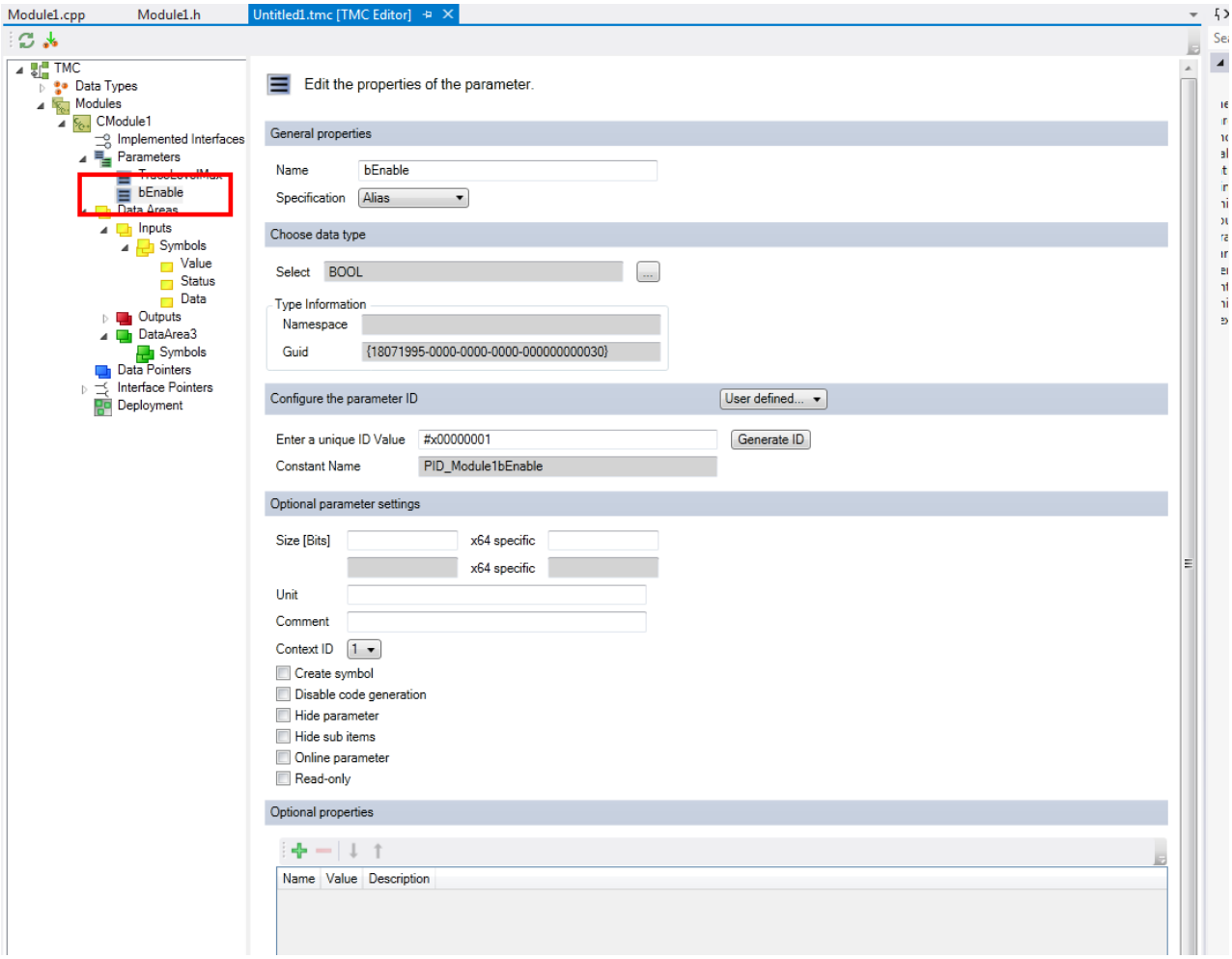
Bedenken Sie die verschiedenen Zustandsübergänge Ihrer Zustandsmaschine [▶ 27]:



Die Parameter werden beim Übergang Init->Preop und gegebenenfalls Preop->Safeop festgelegt.

11.3.4.2.2 Parametereigenschaften

Parameter properties: Die Eigenschaften des Parameters bearbeiten



Allgemeine Eigenschaften

Name: Name der Schnittstelle

Specification: Datentyp des Parameters. Siehe Spezifikation.

Datentyp auswählen

Select: Datentyp auswählen

Typinformation

- **Namespace:** Benutzerdefinierter Namensraum des Datentyps
- **GUID**: Eindeutige ID des Datentyps

Enter a unique ID Value: Einen eindeutigen ID-Wert eingeben. Siehe Parameter [▶ 93].

Constant Name: Quellcodename der Parameter-ID

Optionale Parametereinstellungen

Size [Bits]: Berechnete Größe in Bits (weiße Felder) und in „Byte.Bit“-Notation (graue Felder). Für x64 wird eine besondere Größenkonfiguration bereitgestellt.

Unit: Eine Einheit der Variablen

Comment: Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist

Context ID: Kontext, welcher beim Zugriff auf den Parameter durch ADS verwendet wird

Create Symbol: Standardeinstellung für ADS-Symbolerstellung

Disable Code Generation: Die Code-Generierung freigeben/sperren

Hide parameter: Umschalten zwischen Parameter ein-/ausblenden in der Systemmanager Ansicht

Hide sub items: Wenn der Datentyp über Unterelemente verfügt, dann wird der Systemmanager keinen Zugriff auf die Unterelemente gewähren. Dies sollte z.B. im Falle großer Arrays verwendet werden.

Online parameter: Als Online-Parameter festlegen

Read-only: Wechseln zu nur Lesezugriff für Systemmanager

Optionale Eigenschaften

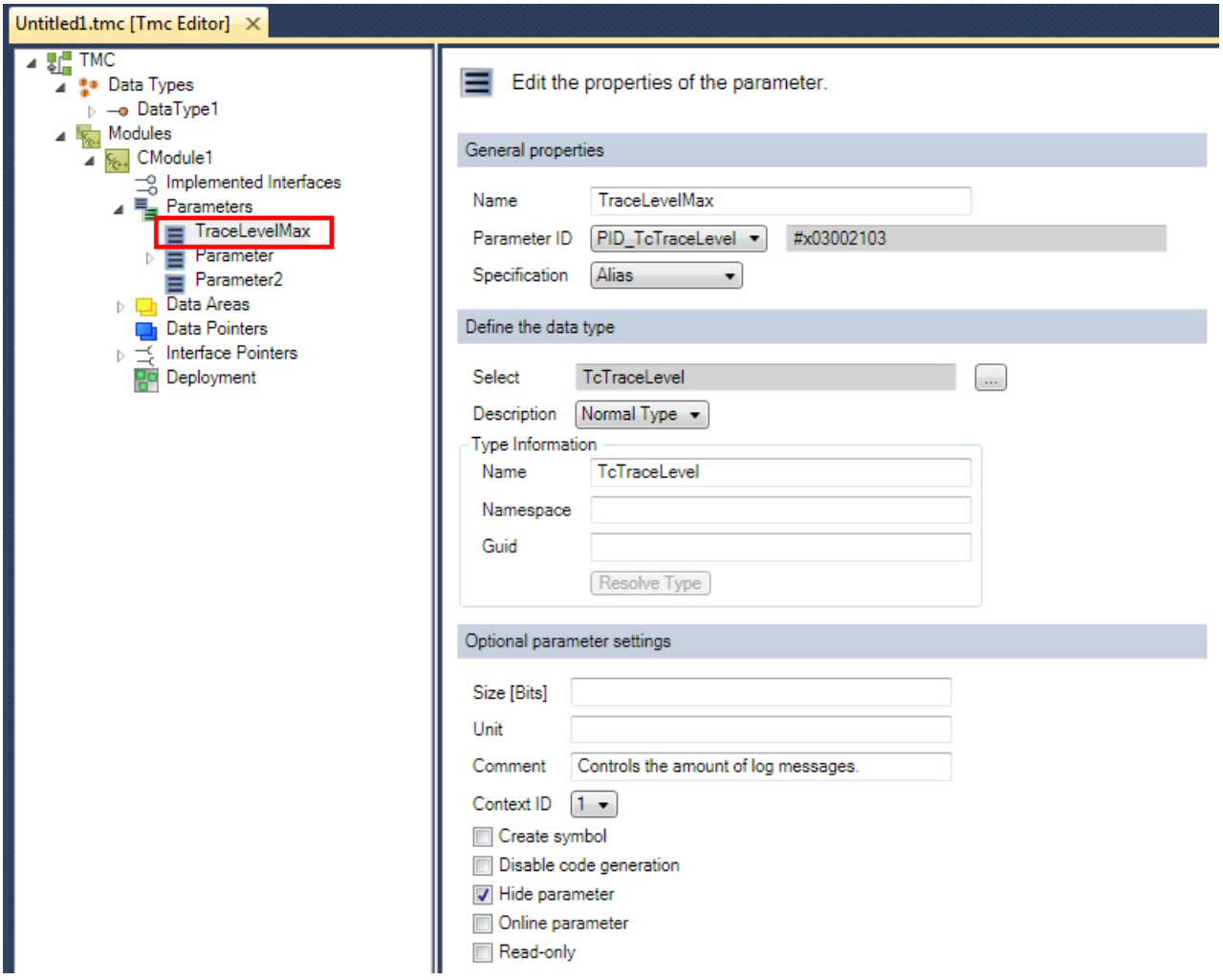
Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Parameters.
Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

11.3.4.2.3 TraceLevelMax

TraceLevelMax: Parameter der den Tracelevel festlegt.
Dies ist ein vordefinierter Parameter, den die meisten TwinCAT-Modulvorlagen bereitstellen (außer die leere TwinCAT-Modulvorlage).

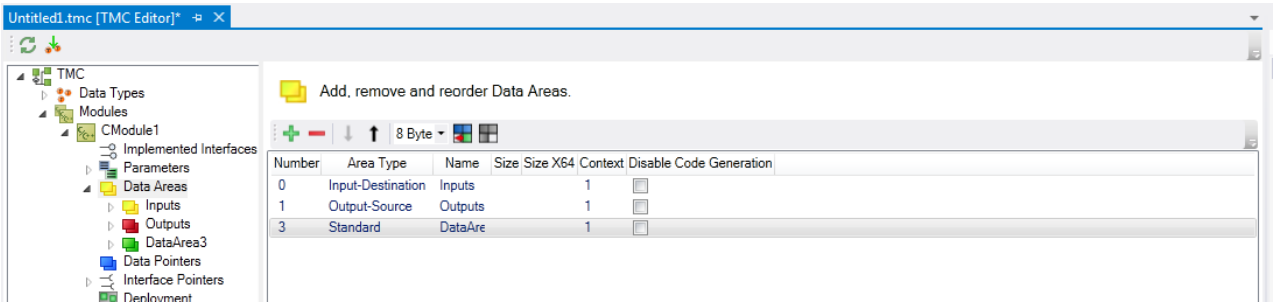
Die Einstellungen dieses Parameters sollten nicht geändert werden.

Siehe Meldungen aufzeichnen [▶ 176]



11.3.4.3 Datenbereiche

Data Areas: Dialog zum Bearbeiten der Datenbereiche Ihres Moduls



Symbol	Funktion
	Einen neuen Datenbereich hinzufügen
	Löscht den ausgewählten Datenbereich
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben
	Byte Alignment auswählen
	Ausgewählten Datentyp ausrichten
	Datenformat des ausgewählten Bereichs zurücksetzen

Hinweis	**Rekursion bei Festlegung eines Alignment** Bei der Festlegung des Alignment eines Datenbereichs wird dieses für alle seine Elemente (Symbole und auch deren Unterelemente) zu Grunde gelegt. Benutzerdefiniertes Alignment wird überschrieben.

Number: Nummer des Datenbereichs

Type: Definiert den Zweck und die Lage des Datenbereichs

Name: Name des Datenbereichs

Size: Größe des Parameters. Für x64 sind andere Größen möglich.

Context: Zeigt die Kontext-ID an

Disable Code Generation: Die Code-Generierung freigeben/sperren

11.3.4.3.1 Datenbereiche und Variablen hinzufügen / bearbeiten / löschen

Mit Hilfe des TwinCAT Module Class (TMC) Editors können Eigenschaften und Funktionalitäten einer TwinCAT-Klasse hinzugefügt, bearbeitet und gelöscht werden.

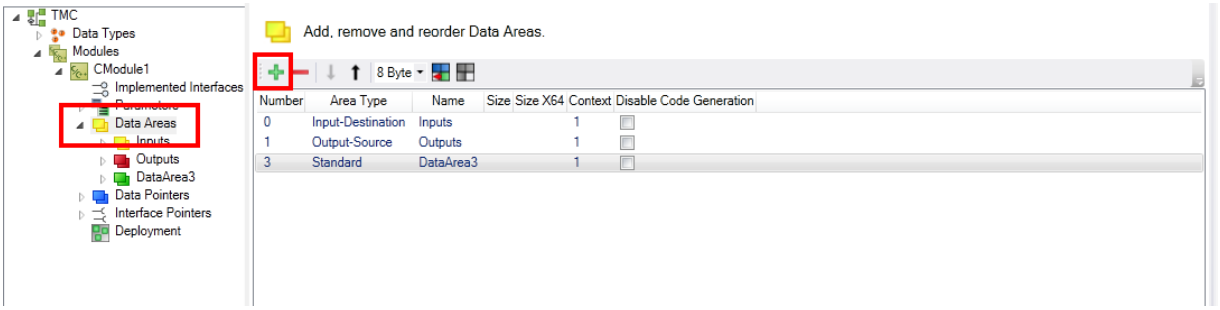
Dieser Artikel beschreibt:

- Einen neuen Datenbereich in der TMC-Datei erstellen
- Neue Variablen in einen Datenbereich erzeugen [▶ 107]

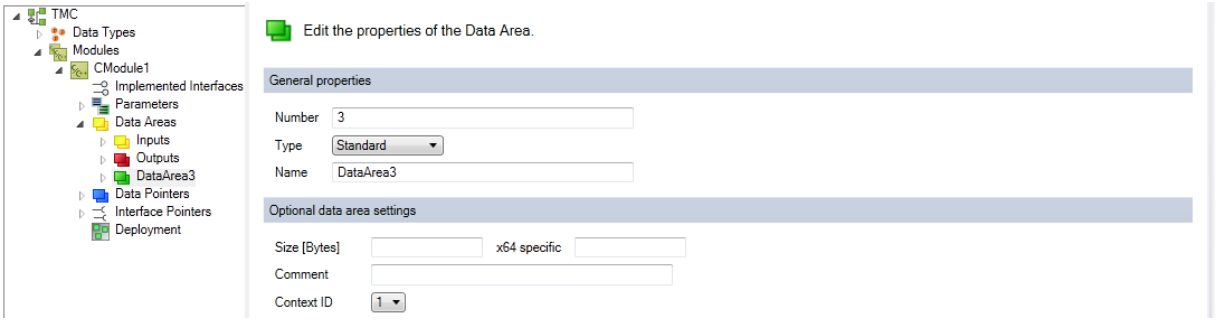
- Z.B. den Namen oder Datentyp [▶ 107] von in der TMC-Datei bestehenden Variablen bearbeiten
- In der TMC-Datei bestehende Variablen löschen [▶ 108]

Einen neuen Datenbereich erzeugen

1. Nach dem Starten des TMC Editors den Knoten „Data Areas“ des Moduls auswählen
2. Durch Klicken auf die + Schaltfläche wird ein neuer Datenbereich erzeugt



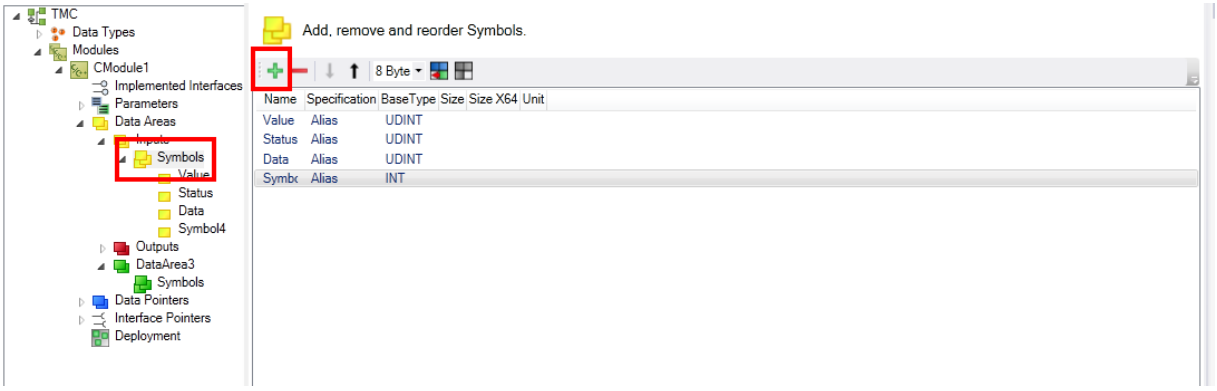
3. Um die Eigenschaften des Datenbereichs zu erhalten, auf die Tabelle doppelklicken oder auf den Knoten klicken



4. Den Datenbereich umbenennen

Eine neue Variable erzeugen

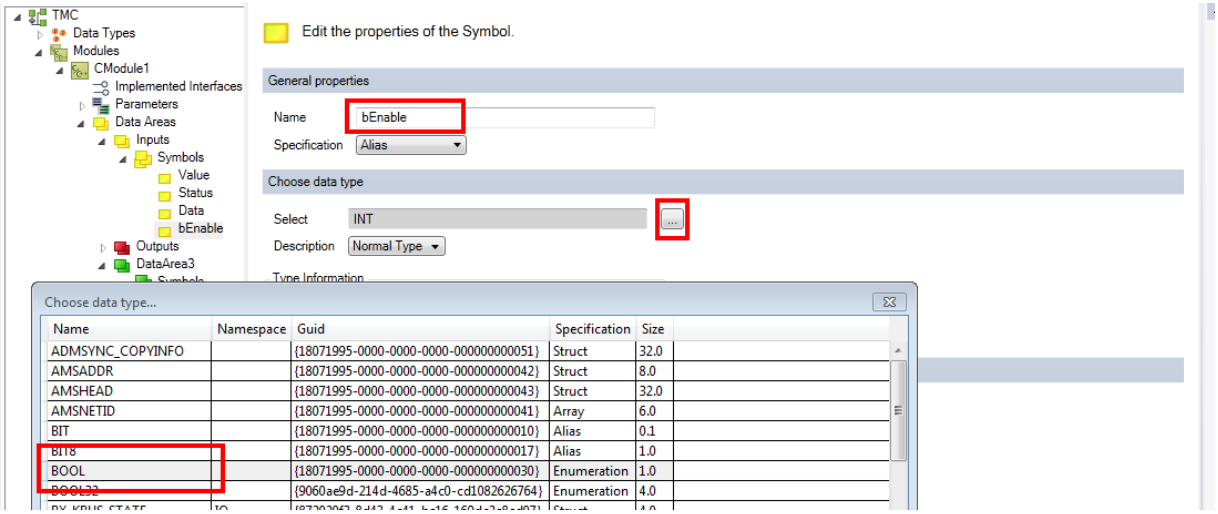
5. Den Unterknoten „Symbols“ des Datenbereichs auswählen.
6. Dieser Datenbereich kann mit einem Klick auf die „+“ Schaltfläche um eine neue Variable erweitert werden. Daraufhin wird „Symbol4“ als neuer Eintrag aufgeführt.



Name oder Datentyp von bestehenden Variablen bearbeiten

7. Wählen Sie den Unterknoten „Symbol4“ oder doppelklicken Sie auf die Zeile. Die Variableneigenschaften werden eingeblendet

8. Geben Sie einen neuen Namen, z.B. „bEnableJob“, ein und ändern den Typ in BOOL

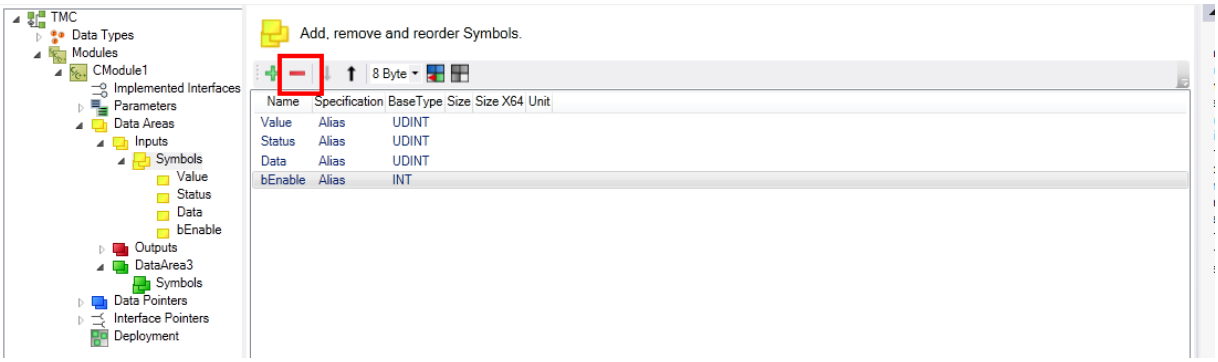


⇨ Letztendlich wird die neue Variable „bEnableJob“ im Datenbereich „Input“ erzeugt.

HINWEIS! Nicht vergessen, den TMC Code Generator erneut auszuführen

Bestehende Variablen löschen

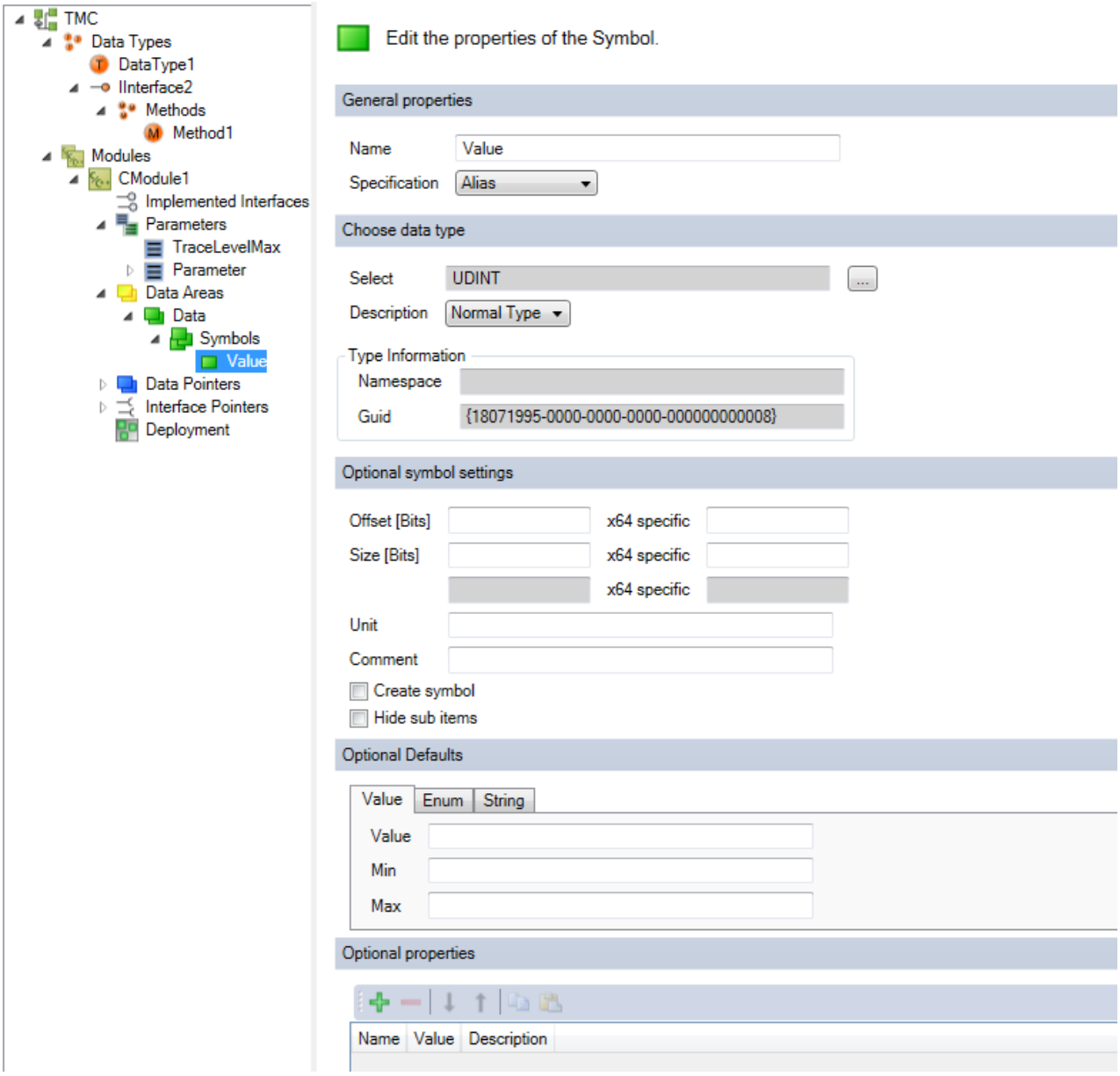
1. Zum Löschen bestehender Variablen im Datenbereich die Variable auswählen und dann auf das Löschen-Symbol klicken:
   In diesem Beispiel „MachineStatus1“ auswählen und auf „Delete symbol“ klicken



2. Den TMC Code Generator erneut ausführen

11.3.4.3.2 Datenbereichseigenschaften

Data Areas Properties: Dialog zum Bearbeiten der Datenbereichseigenschaften



Allgemeine Eigenschaften

Number: Nummer des Datenbereichs

Type: Definiert den Zweck und die Lage des Datenbereichs Verfügbar sind:

Verknüpfbare Datenbereiche im Systemmanager:

- Input-Source
- Input-Destination
- Output-Source
- Output-Destination

Weitere Datenbereiche:

- Standard (sichtbar aber nicht verknüpfbar im Systemmanager)
- Internal (über ADS erreichbare, aber im Systemmanager nicht sichtbare Symbole)
- MArea (für internen Gebrauch)
- Retain (für internen Gebrauch)
- Not specified (gleich Standard)

Name: Name des Datenbereichs

Optionale Parametereinstellungen

Size [Bytes]: Größe in Byte. Für x64 wird eine besondere Größenkonfiguration bereitgestellt

Comment: Optionaler Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist

Context ID: Kontext-ID aller Symbole dieses Datenbereichs. Wird für die Bestimmung des Mappings verwendet.

Data type name: Wenn angegeben, wird ein Datentyp mit dem angegebenen Namen im Typsystem erstellt

Create Symbol: Standardeinstellung für ADS-Symbolerstellung

Disable Code Generation: Die Code-Generierung freigeben/sperren

Optionale Standardeinstellungen

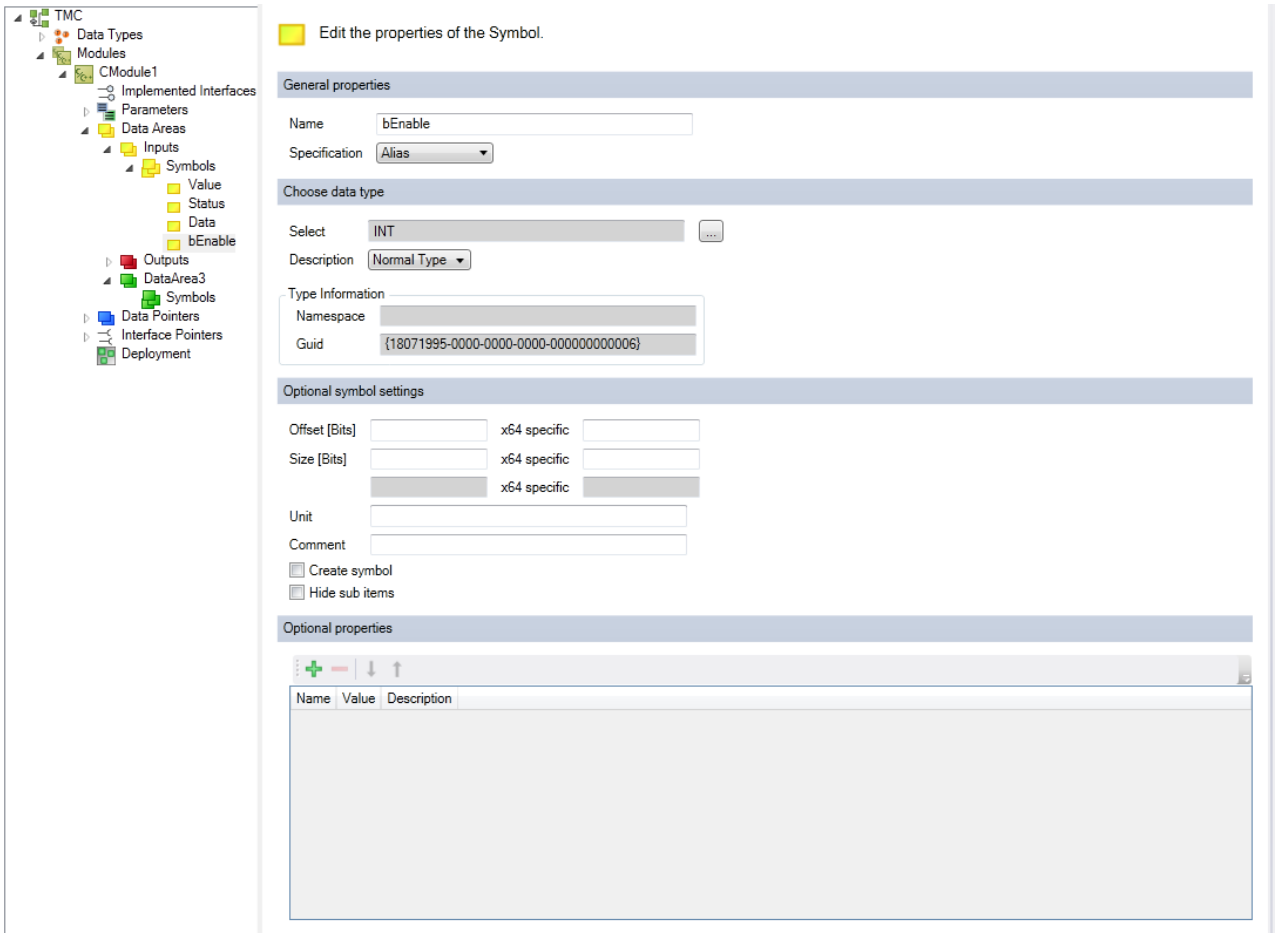
Die Standardeinstellungen können in Funktion des Datentyps definiert werden.

Optionale Eigenschaften

Eine aus Name, Wert und Beschreibung bestehende Tabelle zwecks Kommentierung des Datenbereichs. Diese Information wird in den TMC- und auch TMI-Dateien bereitgestellt.
Diese Eigenschaften können sowohl von TwinCAT-Funktionen als auch von benutzerdefinierten Programmen verwendet werden.

11.3.4.3.3 Symboleigenschaften

Symbols: Dialog für die Bearbeitung der Symbole des Datenbereichs



Allgemeine Eigenschaften

Name: Name des Symbols

Specification: Datentyp des Symbols, siehe Datentypeigenschaften [▶ 84]

Datentyp auswählen

Select: Datentyp auswählen - hierbei kann es sich um Basisdatentypen von TwinCAT oder um benutzerdefinierte Datentypen handeln.

Description: Festlegen, ob der Typ folgendes ist:

- Normaler Typ
- Zeiger
- Zeiger auf Zeiger
- Zeiger auf Zeiger auf Zeiger
- eine Referenz

Typinformation

- **Namespace:** Namensraum für ausgewählten Datentyp
- **GUID**: Eindeutige ID des Datentyps

Optionale Datentypeinstellungen

Offset [Bits]: Offset des Symbols innerhalb des Datenbereichs. Für x64-Plattform kann anderes Offset festgelegt werden.

Size [Bits]: Größe in Bits, falls angegeben. Für x64-Plattform kann eine andere Größe festgelegt werden.

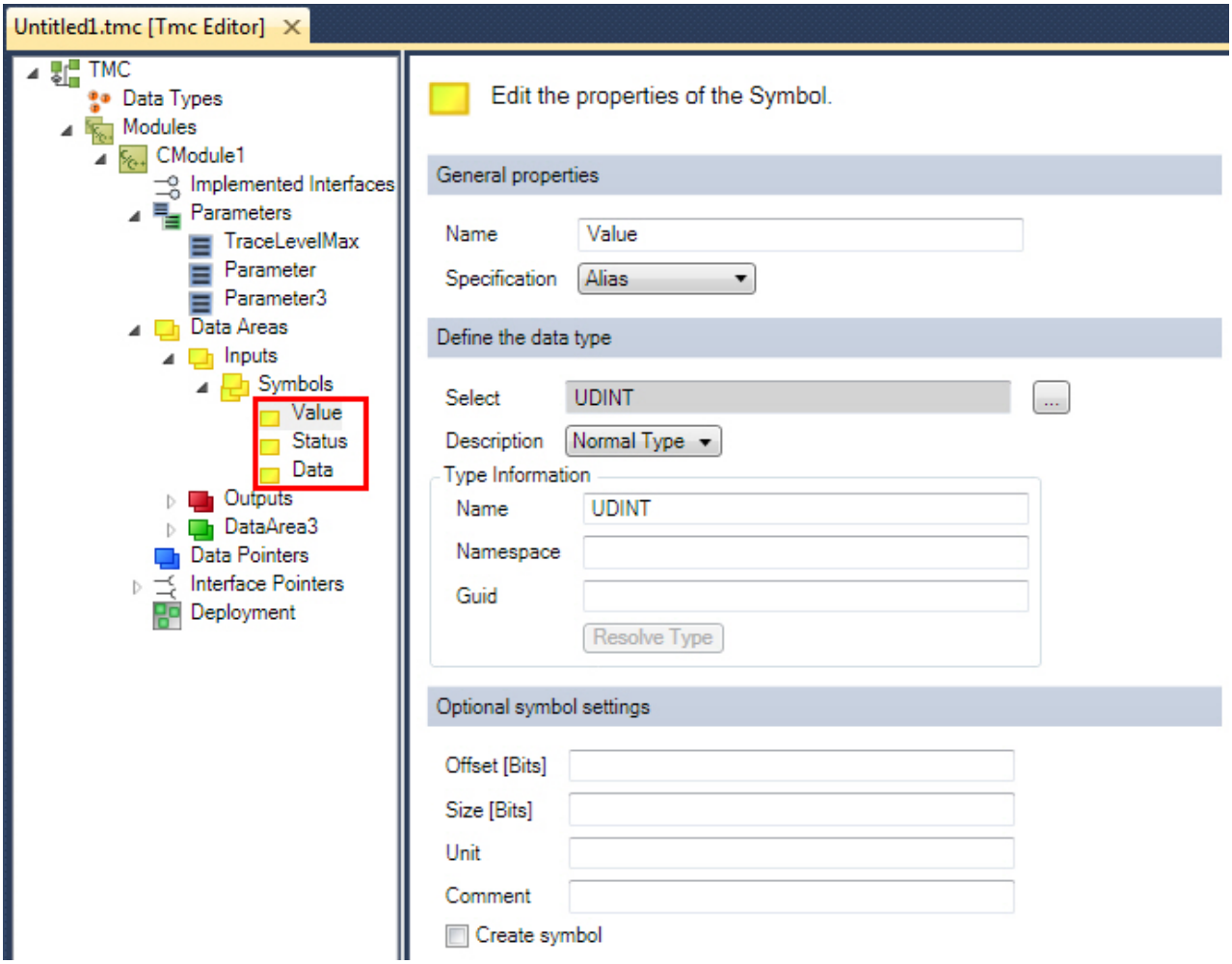
Comment: Optionaler Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist

Create Symbol: Standardeinstellung für ADS-Symbolerstellung

Hide sub items: Wenn Variable über Unterelemente verfügt, dann wird der Systemmanager keinen Zugriff auf die Unterelemente gewähren. Dies sollte z.B. im Falle großer Arrays verwendet werden.

11.3.4.3.3.1 TwinCAT Module Class Editor - Data Areas Symbols Properties

Data Areas Symbols Properties: Dialog zum Bearbeiten der Datenbereichssymboleigenschaften



Allgemeine Eigenschaften

Name: Name der Schnittstelle

Specification: Datentyp des Parameters

Verfügbare Spezifikationen sind:

- **Alias**: Ein Alias eines Standarddatentyps (z.B. INT) erzeugen
- **Array:** Ein benutzerdefiniertes Array erstellen
- **Enumeration:** Eine benutzerdefinierte Aufzählung erstellen
- **Struct:** Eine benutzerdefinierte Struktur erstellen
- **Interface:** Eine neue Schnittstelle erzeugen

Den Datentyp definieren

Select: Datentyp auswählen

Description: Beschreibung festlegen

Typinformation

Name: Name des ausgewählten Standardtyps

Namespace: Benutzerdefinierter Namensraum des Datentyps

GUID: Eindeutige ID des Datentyps

Optionale Datentypeinstellungen

Offset [Bits]: Speicher-Offset

Size [Bits]: Berechnete Größe in Bits

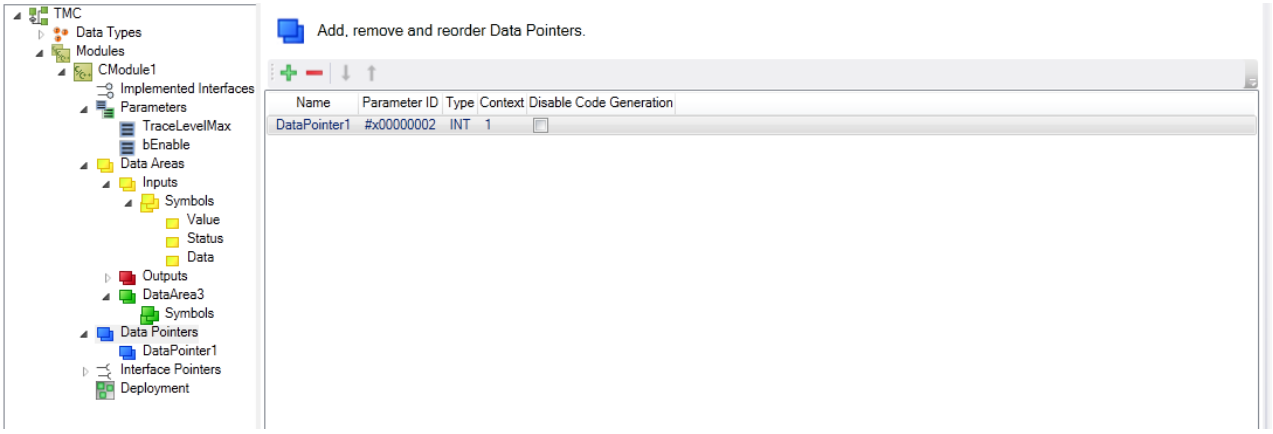
Unit: Optional

Comment: Optional

Create symbol: Standardeinstellung für ADS-Symbolerstellung

11.3.4.4 Datenzeiger

Data Pointer: Dialog zum Bearbeiten der Datenzeiger Ihres Moduls



Symbol	Funktion
	Einen neuen Datenzeiger hinzufügen
	Löscht den ausgewählten Datenzeiger
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben

Name: Name des Datenzeigers

Parameter ID: Eindeutige ID des Parameters

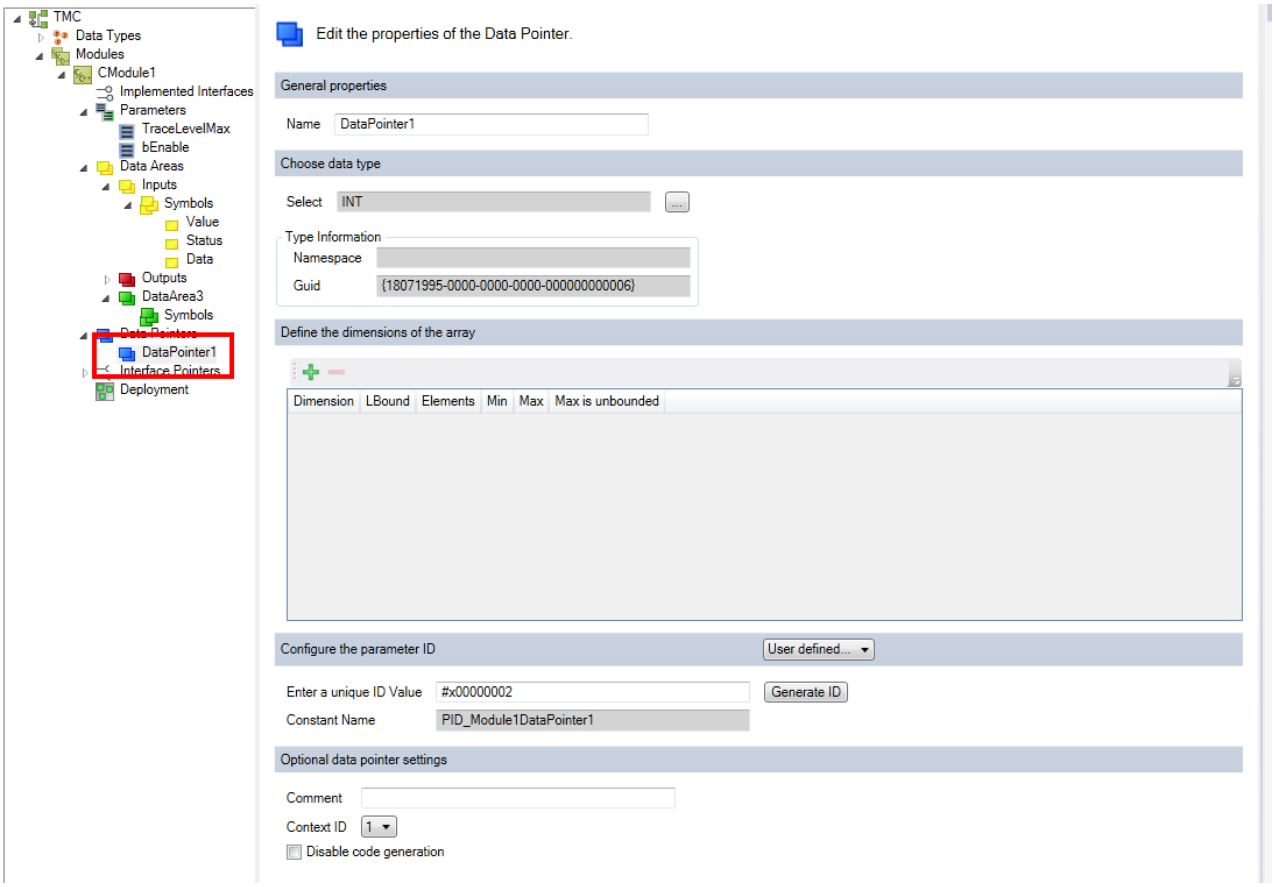
Type: Definiert den Zeigertyp

Context: Zeigt die Kontext-ID an

Disable Code Generation: Die Code-Generierung freigeben/sperren

11.3.4.4.1 Datenzeigereigenschaften

Data Pointer Properties: Die Eigenschaften des Datenzeigers bearbeiten



Allgemeine Eigenschaften

Name: Name des Datenzeigers

Den Datentyp definieren

Select: Datentyp auswählen

Typinformation

- **Name**: Name des ausgewählten Datentyps
- **GUID**: Eindeutige ID des Datentyps

Die Dimension des Array definieren

Siehe hier [▶ 87].

Die Parameter-ID konfigurieren

Enter a unique ID Value: Einen eindeutigen ID-Wert eingeben. Siehe Parameter [▶ 93].

Constant Name: Quellcodename der Parameter-ID

Optionale Datenzeigereinstellungen

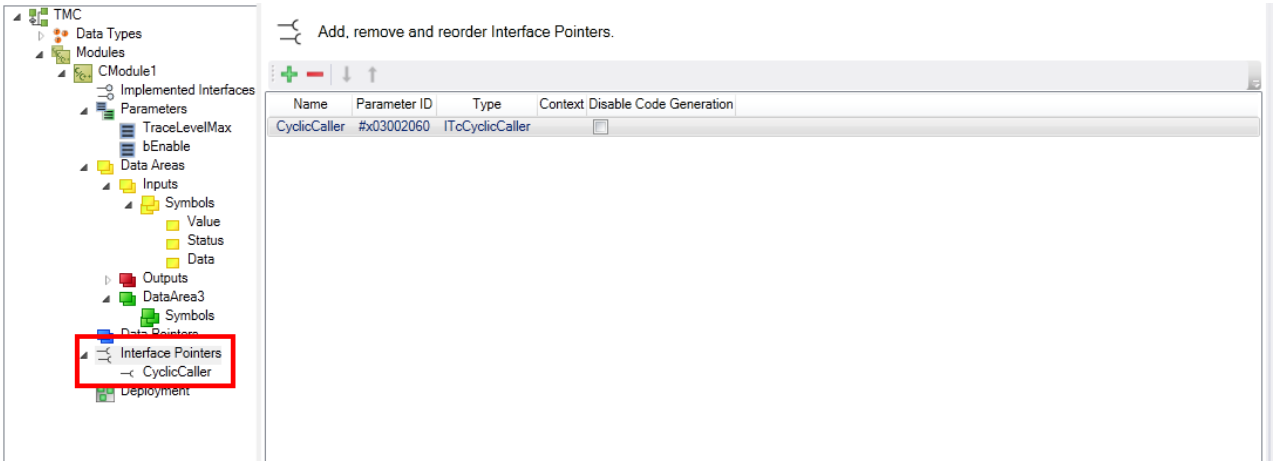
Comment: Kommentar, der z.B. im Instanzenkonfigurator sichtbar ist

Context ID: Kontext-ID des Datenzeigers

Disable Code Generation: Die Code-Generierung freigeben/sperren

11.3.4.5 Schnittstellenzeiger

Interface Pointers: Schnittstellenzeiger hinzufügen, entfernen und neu ordnen



Symbol	Funktion
	Schnittstellenzeiger hinzufügen
	Löscht den ausgewählten Zeiger
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben

Name: Name der Schnittstelle

Parameter ID: Eindeutige ID des Schnittstellenzeigers

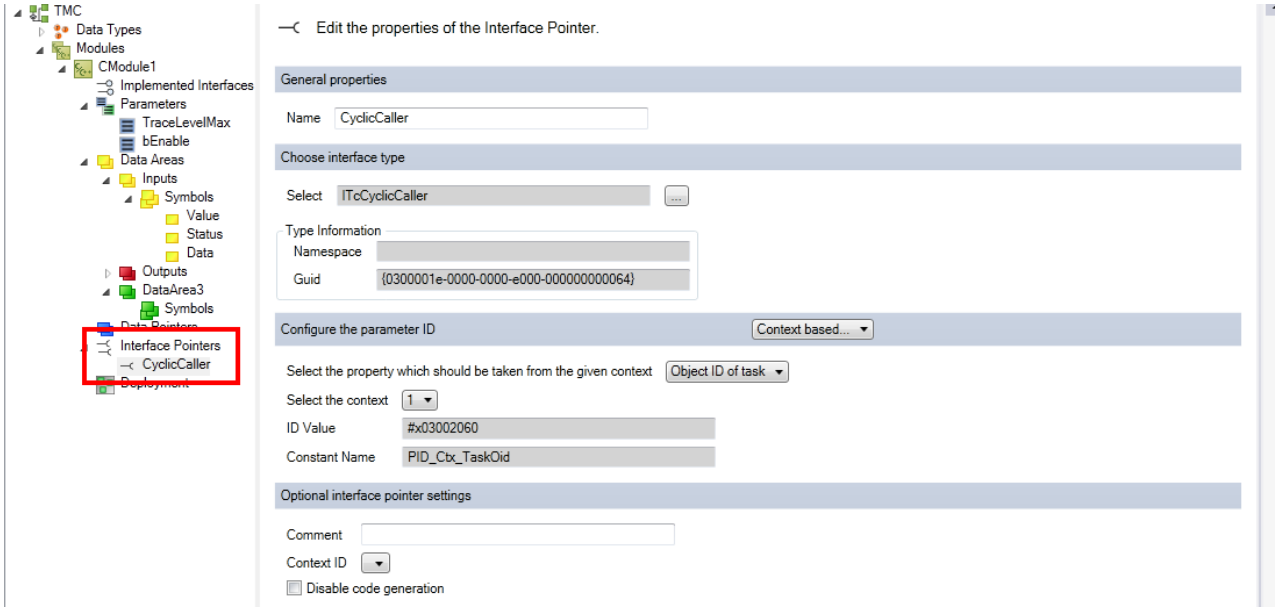
Type: Typ des Schnittstellenzeigers

Context: Kontext der Schnittstelle

Disable Code Generation: Die Code-Generierung freigeben/sperren

11.3.4.5.1 Eigenschaften des Schnittstellenzeigers

Interface Pointer Properties: Die Eigenschaften des Schnittstellenzeigers bearbeiten



Allgemeine Eigenschaften

Name: Name des Schnittstellenzeigers

Basisschnittstelle auswählen

Select: Auswahl der Schnittstelle

Typinformation

- **Namespace:** Namensraum der Schnittstelle
- **GUID**: Eindeutige ID der Schnittstelle

Die Parameter-ID konfigurieren

Siehe Parameter [▶ 93].

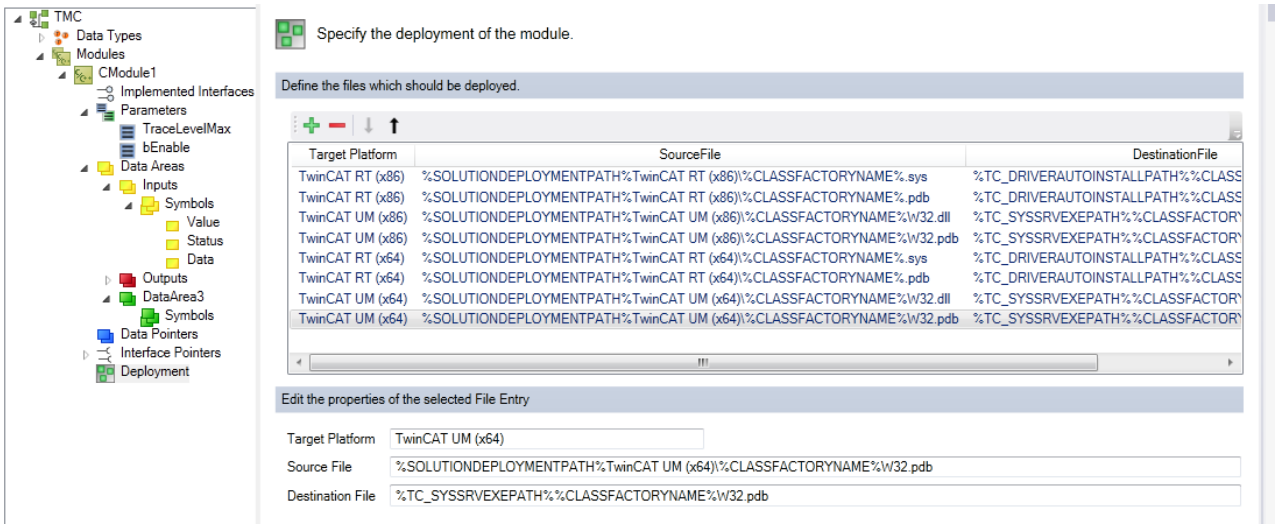
Comment: Optional

Context ID: Kontext-ID des Schnittstellenzeigers

Disable Code Generation: Die Code-Generierung freigeben/sperren

11.3.4.6 Bereitstellung

Deployment: Speicherorte für die bereitgestellten Module festlegen



Symbol	Funktion
	Einen neuen Dateieintrag hinzufügen
	Einen Dateieintrag löschen
	Verschiebt das ausgewählte Element um eine Position nach unten
	Verschiebt das ausgewählte Element um eine Position nach oben

Target Platform: Legt die Plattform und den Laufzeitkontext (x86 / x64) fest.

Source File: Pfad zu den Quelldateien

Destination File: Pfad zu den Binärdateien

11.4 TwinCAT Module Instance Configurator

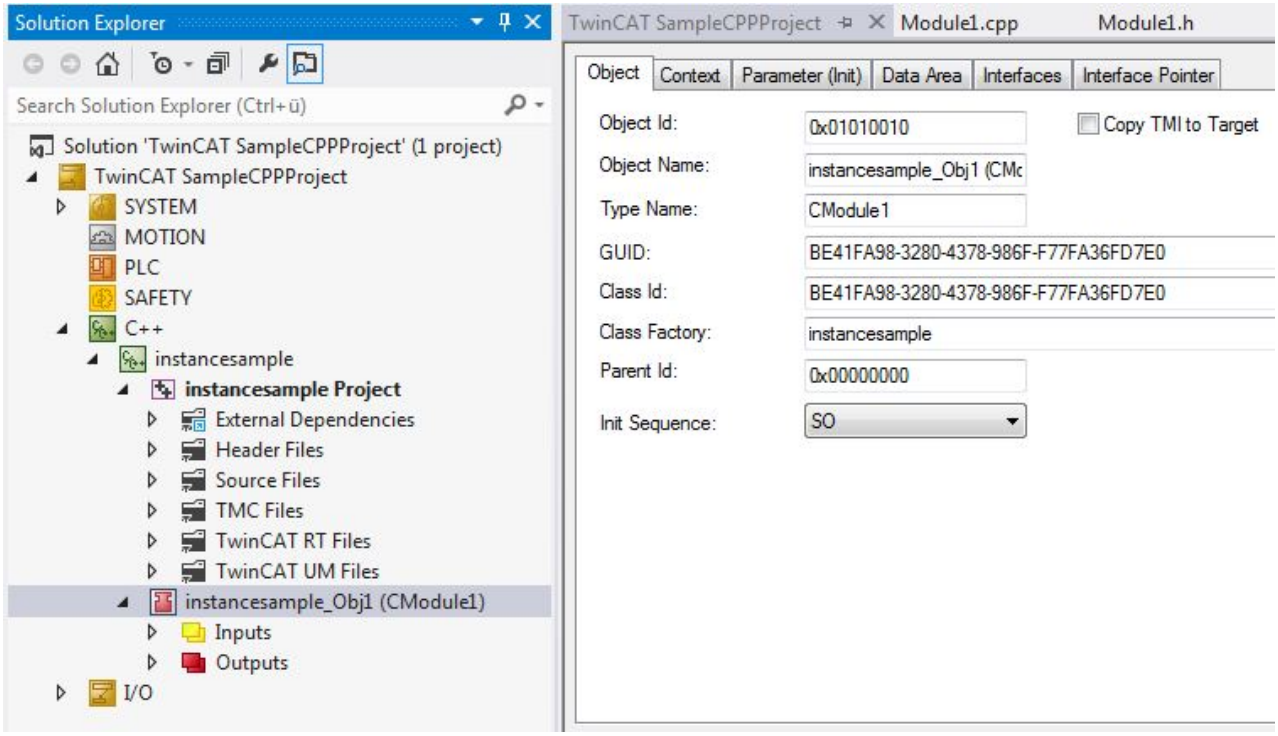
Der oben beschriebene TwinCAT 3 Module Class (TMC) Editor definiert Treiber auf Klassenebene. Diese werden instanziiert und müssen über den TwinCAT 3 Instance Configurator konfiguriert werden.

Die Konfiguration betrifft z.B. den Kontext (einschließlich Task, die das Modul aufruft), Parameter und Zeiger.

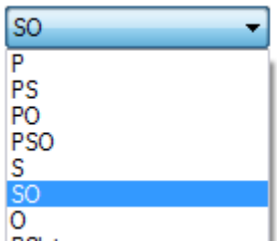
Instanzen von C++ Klassen werden durch Rechtsklick auf den C++ Projektordner erstellt, siehe Schnellstart. In diesem Kapitel wird die Konfiguration dieser Instanzen ausführlich beschrieben.

Durch Doppelklick auf die generierte Instanz wird der Konfigurationsdialog mit mehreren Fenstern geöffnet.

11.4.1 Objekt



- **Object Id:** Die Objekt-ID, die für die Identifizierung dieser Instanz im TwinCAT System herangezogen wird.
- **Object Name:** Name des Objekts, der für die Darstellung der Instanz im Solution Explorer-Baum verwendet wird.
- **Type Name:** Typinformation (Klassenname) der Instanz
- **GUID:** Modulklassen-GUID
- **Class Id:** Klassen-ID der Implementierungsklasse (normalerweise sind GUID und ClassId identisch)
- **Class Factory:** Verweist auf den Treiber, der die Class Factory bereitstellt, welche für die Erstellung einer Instanz des Moduls verwendet wurde.
- **Parent Id:** Beinhaltet die ObjectID des Parent, falls vorhanden.
- **Init Sequence:** Legt die Initialisierungszustände für die Bestimmung des Startup-Verhaltens der interagierenden Module fest. Siehe hier [▶ 27] für genaue Beschreibung der Zustandsmaschine.



Festlegung des Startup-Verhaltens von mehreren TcCOM Instanzen.

TcCOM Instanzen können sich gegenseitig auf einander beziehen - z.B. zwecks Wechselwirkung über Daten- oder Schnittstellenzeiger. Zur Bestimmung des Startup-Verhaltens legt die „Init Sequenz“ von jeder TcCOM Instanz zu „haltende“ Zustände für alle übrigen Module fest.

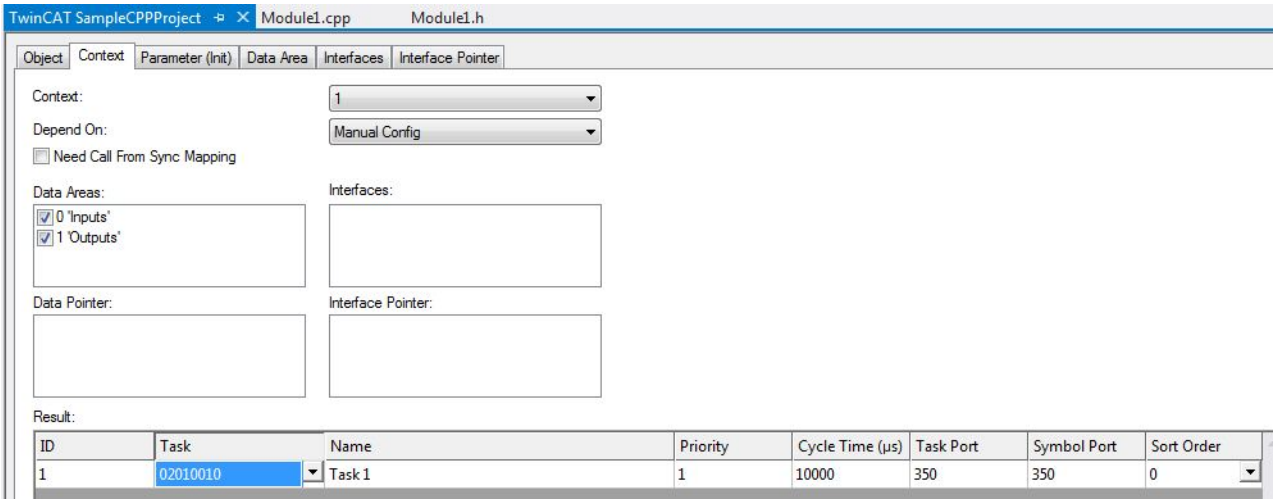
Der Name einer Init Sequenz besteht aus dem Kurznamen der TcCOM Zustandsmaschine. Wenn der Kurzname eines Zustands (I,P,S,O) im Namen der Init Sequenz enthalten ist, dann werden die Module in diesem Zustand warten, bis alle anderen Module zumindest diesen Zustand erreicht haben. Beim nächsten Übergang kann das Modul sich auf alle anderen Modulinstanzen beziehen, um sich mindestens in diesem Zustand zu befinden.

Wenn z.B. ein Modul die Init Sequenz „PS“ aufweist, werden die IP-Übergänge aller anderen Module ausgeführt, sodass alle Module sich im Zustand „Preop“ befinden.

Anschließend wird der PS-Übergang des Moduls ausgeführt und das Modul kann sich darauf verlassen, dass die anderen Module sich im „Preop“-Zustand befinden.

- **Copy TMI to target:** Die TMI (TwinCAT Module Instance) Datei generieren und an das Ziel übergeben.

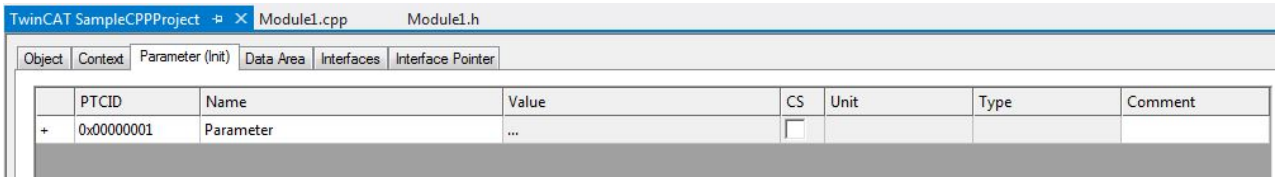
11.4.2 Kontext



- **Context:** Den zu konfigurierenden Kontext auswählen (siehe TMC Editor für Hinzufügen verschiedener Kontexte).

  HINWEIS! Ein Datenbereich ist einem Kontext zugeordnet
- **Data Areas / Interfaces / Data Pointer und Interface Pointer:** Jede Instanz kann so konfiguriert werden, dass sie in TMC definierte Elemente hat oder nicht.
- **Result Table:** Liest der IDs, die konfiguriert werden müssen. Zumindest muss der Kontext („Task“-Spalte) dem Task entsprechend konfiguriert werden.

11.4.3 Parameter (Init)

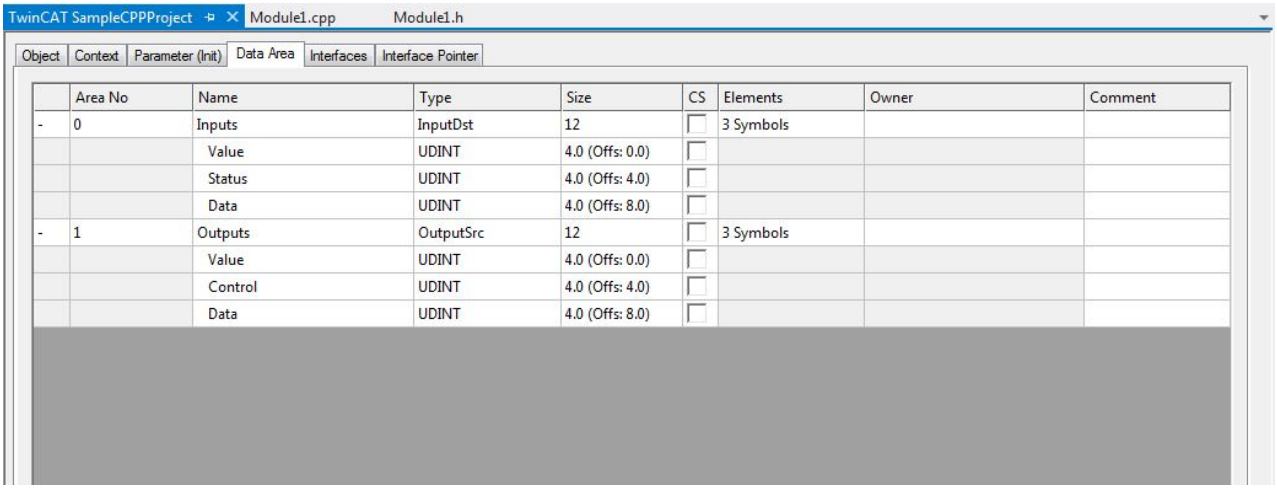


Die Liste aller Parameter (wie in TMC definiert) kann für jede Instanz mit Werten initialisiert werden.

Spezielle ParameterIDs (PTCID) werden verwendet, um die Werte automatisch festzulegen. Diese werden mit Hilfe des Parameterdialogfensters des TMCEditor's wie hier [▶ 93] beschrieben konfiguriert.

Die CS (CreateSymbol) Checkbox erzeugt das ADS-Symbol für jeden Parameter, demzufolge ist er von außen erreichbar.

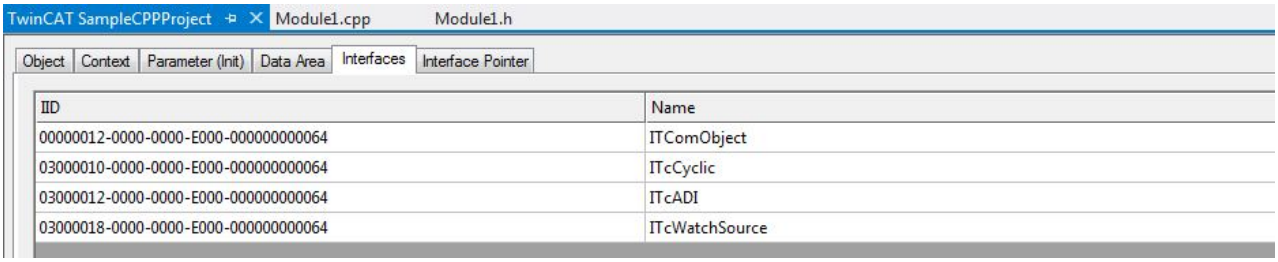
11.4.4 Data Area



Liste aller Datenbereiche und ihrer Variablen (wie in TMC definiert).

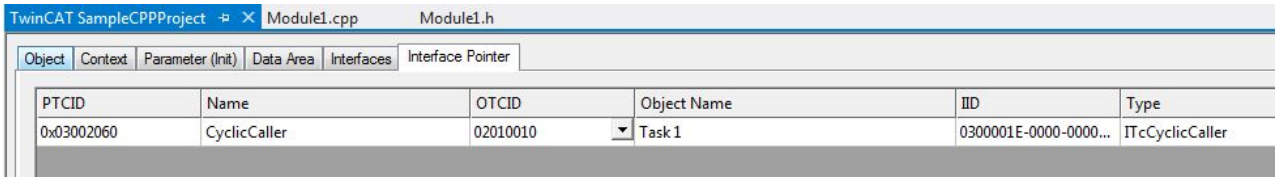
Die CS (CreateSymbol) Checkbox erzeugt das ADS-Symbol für jeden Parameter, so ist die Variable von außen erreichbar.

11.4.5 Schnittstellen



Liste aller implementierter Schnittstellen (wie in TMC definiert).

11.4.6 Schnittstellenzeiger

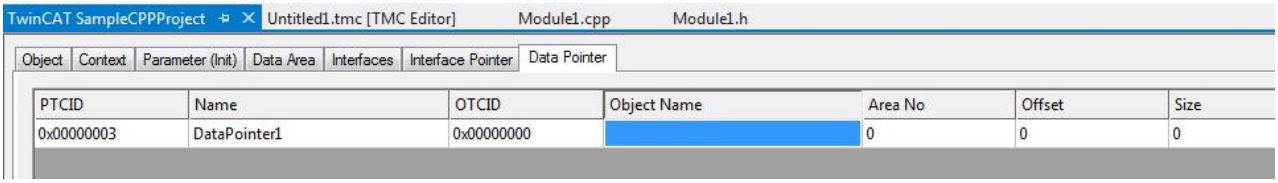


Liste aller Schnittstellenzeiger (wie in TMC definiert).

Spezielle ParameterIDs (PTCID) werden verwendet, um die Werte automatisch festzulegen. Diese werden mit Hilfe des Parameterdialogfensters des TMCEditor's wie hier [▶ 93] beschrieben konfiguriert.

Die OTCID-Spalte definiert die Zeiger auf die Instanz, die zu verwenden ist.

11.4.7 Datenzeiger



Liste aller Datenzeiger (wie in TMC definiert).

Spezielle ParameterIDs (PTCID) werden verwendet, um die Werte automatisch festzulegen. Diese werden mit Hilfe des Parameterdialogfensters des TMCEditor's wie hier [▶ 93] beschrieben konfiguriert.

Die OTCID-Spalte definiert die Zeiger auf die Instanz, die zu verwenden ist.

12 Programmierreferenz

TwinCAT bietet eine Vielzahl Basisfunktionen. Sie alle können für einen TwinCAT C++ Programmierer sehr nützlich sein und werden hier dokumentiert.

Es besteht eine Vielzahl an C++ Beispielen [▶ 196], die unbedingt angeschaut werden sollten.

12.1 Dateibeschreibung

Bei der Entwicklung von TwinCAT C++ Modulen ist ein direkter Umgang mit Dateien des Dateisystems möglich. Dies ist von Interesse, entweder um zu verstehen wie das System funktioniert oder für spezielle Anwendungsfälle, wie z.B. manuelle Dateiübertragung, usw.

Hier eine Liste der Dateien, die in Zusammenhang mit C++ Modulen stehen.

Datei	Beschreibung	Weitere Informationen
Engineering / XAE		
*.sln	Visual Studio Solution-Datei, beherbergt TwinCAT- und Nicht-TwinCAT-Projekte	
*.tsproj	TwinCAT Projekt, Sammlung aller verschachtelten TwinCAT-Projekte, wie TwinCAT C++ oder TwinCAT SPS-Projekt	
_Config/	Ordner enthält weitere Konfigurationsdateien (*.xti), die zum TwinCAT-Projekt gehören.	Siehe Menü Tools| Options| TwinCAT| XAE-Environment| File Settings
_Deployment/	Ordner für kompilierte TwinCAT C++ Treiber	
*.tmc	TwinCAT Module Class Datei (XML-basiert)	Siehe TwinCAT Module Class Editor (TMC) [▶ 69]
*.rc	Resourcendatei	Siehe Siehe Version/ Herstellerinformation [▶ 188]
.vcxproj.	Visual Studio C++ Projektdateien	
*ClassFactory.cpp/.h	Class Factory für diesen TwinCAT Treiber	
*Ctrl.cpp/.h	Treiber hochladen und entfernen für TwinCAT UM Plattform	
*Driver.cpp/.h	Treiber hochladen und entfernen für TwinCAT RT Plattform	
*Interfaces.cpp/.h	Deklaration der TwinCAT COM Schnittstellenklassen	
*W32.cpp./.def/.idl		
*.cpp/.h	Eine C++/Header-Datei pro TwinCAT Modul im Treiber. Benutzercode hier einfügen.	
Resource.h	Wird von *.rc Datei benötigt	
TcPch.cpp/.h	Wird für die Erstellung von vorkompiliertem Header verwendet	
%TC_INSTALLPATH% \CustomConfig\Modules*	Veröffentlichtes TwinCAT Treiberpaket normalerweise C:\TwinCAT\3.x\CustomConfig \Modules*	Siehe Module exportieren [▶ 31]
Laufzeit / XAR		
%TC_BOOTPRJPATH% \CurrentConfig*	Derzeitige Konfigurationssetup normalerweise C:\TwinCAT\3.x\Boot	
%TC_DRIVERAUTOINSTALLPATH% *.sys/pdb	Kompilierter, plattformspezifischer Treiber • C:\TwinCAT\3.x\Driver\AutoInstall (System geladen) • C:\TwinCAT\3.x\Driver\AutoLoad (TcLoader geladen)	
%TC_BOOTPRJPATH% \TM \OBJECTID.tmi	TwinCAT Module Instance Datei Beschreibt Variablen des Treibers Dateiname lautet „ObjectID.tmi“ Normalerweise C:\TwinCAT\3.x\Boot\TMI \OTCID.tmi	
Temporäre Dateien		
*.sdf	IntelliSense Datenbank	
*.suo / *.v12.suo	Benutzerspezifische und Visual Studio spezifische Dateien	
*.tsproj.bak	Automatisch generierte Sicherungsdatei von tsproj	

Datei	Beschreibung	Weitere Informationen
ipch/	Für vorkompilierter Header erstelltes Zwischen-Verzeichnis	

12.2 Limitierungen

TwinCAT 3 C++ Module [▶ 23] werden im Windows Kernel-Modus ausgeführt. Deswegen müssen die Entwickler einige Limitierungen beachten:

- Win32 API ist nicht im Kernel-Modus verfügbar. (siehe unten [▶ 119])
- Benutzermodusspezifische Bibliotheken (DLL) können nicht verwendet werden. (siehe Bibliotheken von Drittanbietern [▶ 191])
- Speicherplatz für dynamische Allozierung mittels des new Operators in Laufzeitkontext ist aufgrund der Größe vom Router-Speicher begrenzt (kann im Verlauf des Engineerings konfiguriert werden).
- Untermenge der C++ Laufzeit wird unterstützt
- C++ Ausnahmen werden nicht unterstützt
- Laufzeit-Typinformation (RTTI, Runtime type Information) wird nicht unterstützt (siehe unten [▶ 119])
- Untermenge von STL wird unterstützt (siehe STL / Container [▶ 175])
- Unterstützung für Funktionen von math.h durch TwinCAT Implementierung (siehe Mathematische Funktionen [▶ 173])

Normalerweise von Win32 API verwendete TwinCAT-Funktionalität

Die originale Win32 API ist nicht im Windows Kernel-Modus verfügbar. Aus diesem Grunde ist hier eine Liste der normalerweise verwendeten Funktionalitäten von Win32 und was stattdessen in TwinCAT verwendet werden kann:

Win32API	TwinCAT Funktionalität
WinSock	TF6311 TCP/UDP Echtzeit
Message Boxen	Tracing [▶ 176]
Datei-I/O	Sieh Schnittstelle ITcFileAccess [▶ 133], Schnittstelle ITcFileAccessAsync [▶ 141] und Beispiel19: Synchroner Dateizugriff [▶ 248], Beispiel20: FileIO-Write [▶ 249], Beispiel20a: FileIO-Cyclic Read / Write [▶ 250]
Synchronisation	Siehe Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an [▶ 247]
Visual C CRT	Siehe RtlR0.h

RTTI dynamic_cast Funktion in TwinCAT

TwinCAT hat keine Unterstützung für dynamic_cast<>.

Stattdessen kann die TCOM-Vorgehensweise möglicherweise verwendet werden. Definieren Sie eine Schnittstelle ICustom, die von ITcUnknown abgeleitet ist und die die Methoden enthält, die von einer abgeleiteten Klasse aufgerufen werden. Die Basisklasse CMyBase wird von ITcUnknown abgeleitet und diese auch implementieren. Die Klasse CMyDerived wird von CMyBase und von ICustom abgeleitet. Sie überschreibt die TcQueryInterface Methode, die dann anstelle von dynamic cast verwendet werden kann.

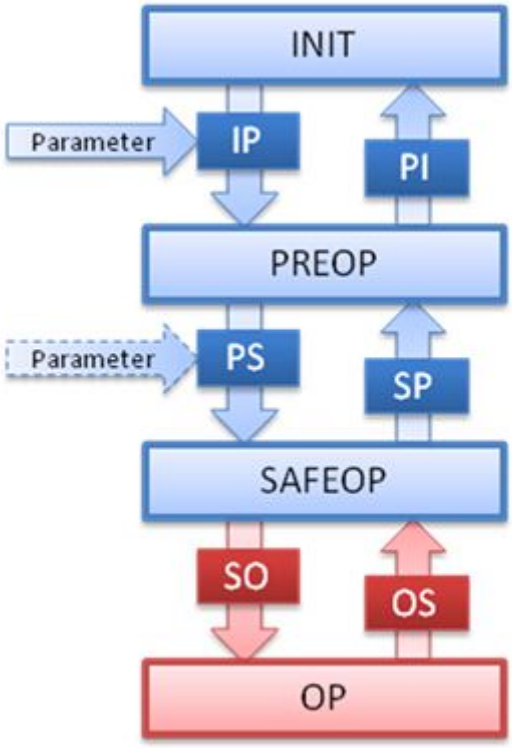
TcQueryInterface kann auch zur Darstellung der IsType() Funktion mittels Auswertung des Rückgabewerts verwendet werden.

Siehe Schnittstelle ITcUnknown [▶ 121]

12.3 Speicherallozierung

Im Allgemeinen wird empfohlen Speicher mit Hilfe von Member-Variablen der Modulklasse zu reservieren. Dies wird automatisch für im TMC Editor definierte Datenbereiche gemacht.
Nichtsdestotrotz besteht die Möglichkeit, Speicherplatz dynamisch zu allozieren. Diese dynamische Zuweisung wird zum Beispiel vom new Operator und den STL / Containern verwendet.

Wird die Speicherallozierung in einem Nicht-Echtzeitkontext vorgenommen, dann wird der Speicher im non-paged Pool des Betriebssystems (blau im Diagramm) alloziert.
Im TwinCAT-Echtzeitkontext wird der Speicher im Routerspeicher (rot im Diagramm) alloziert.



12.4 Schnittstellen

Für die Interaktion der vom Benutzer entwickelten Module mit dem TwinCAT 3 System stehen etliche Schnittstellen zur Verfügung. Auf diesen Seiten werden diese (auf API Ebene) ausführlich beschrieben.

Name	Beschreibung
ITComObject [▶ 123]	Die ITComObject Schnittstelle wird von jedem TwinCAT Modul implementiert.
ITcCyclic [▶ 130]	Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus aufgerufen werden.
ITcCyclicCaller [▶ 131]	Schnittstelle zum Anmelden oder Abmelden der ITcCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.
ITcFileAccess [▶ 133]	Schnittstelle zum Zugriff auf das Dateisystem
ITcFileAccessAsync [▶ 141]	Asynchroner Zugriff auf Dateioperationen.
ITcPostCyclic [▶ 143]	Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus im Anschluss an die Ausgang-Aktualisierung aufgerufen werden.
ITcPostCyclicCaller [▶ 144]	Schnittstelle zum Anmelden oder Abmelden der ITcPostCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.
ITcIoCyclic [▶ 146]	Diese Schnittstelle wird von TwinCAT Modulen implementiert, die bei Eingang-Aktualisierung und bei Ausgang-Aktualisierung innerhalb eines Taskzyklus aufgerufen werden.
ITcIoCyclicCaller [▶ 147]	Schnittstelle zum Anmelden oder Abmelden der ITcIoCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.
ITcRTimeTask [▶ 149]	Abfrage von erweiterten TwinCAT Taskinformationen.
ITcTask [▶ 150]	Abfrage von Zeitstempel und taskspezifischen Informationen eines TwinCAT Tasks.

Darüber hinaus können von TwinCAT 3 bereit gestellte mathematische Funktionen [▶ 173] verwendet werden.

12.4.1 Schnittstelle ITcUnknown

ITcUnknown definiert die Referenzzählung, sowie das Abfragen einer Referenz auf eine spezifischere Schnittstelle.

Syntax

```
TCOM_DECL_INTERFACE("00000001-0000-0000-e000-000000000064", ITcUnknown)
```

Deklariert in: TcInterfaces.h

Benötigtes include: -

Methoden

Sym-bol	Name	Beschreibung
	TcAddRef [▶ 122]	Inkrementiert den Referenzzähler.
	TcQueryInterface [▶ 122]	Abfrage der Referenz an eine implementierte Schnittstelle über der IID
	TcRelease [▶ 123]	Dekrementiert den Referenzzähler.

Anmerkungen

Jede TcCOM Schnittstelle ist direkt oder indirekt von ITcUnknown abgeleitet. Demzufolge implementiert jede TcCOM Modulklasse ITcUnknown, weil sie von ITComObject abgeleitet ist.

Die standardmäßige Implementierung von ITcUnknown sorgt dafür, dass das Objekt nach Freigabe der letzten Referenz gelöscht wird. Aus diesem Grunde muss ein Schnittstellenzeiger nach dem Aufruf von TcRelease() nicht dereferenziert werden.

12.4.1.1 Methode ITcUnknown:TcAddRef

Diese Methode inkrementiert den Referenzzähler.

Syntax

```
ULONG TcAddRef( )
```

Rückgabewert

Daraus resultierender Referenzzählwert.

Beschreibung

Inkrementiert den Referenzzähler und gibt den neuen Wert zurück.

12.4.1.2 Methode ITcUnknown:QueryInterface

Abfrage der Referenz an eine implementierte Schnittstelle über der IID

Syntax

```
HRESULT QueryInterface(RITCID iid, PPVOID pipItf )
```

iid: (Typ: RITCID) Schnittstelle IID

pipItf: (Typ PVOID) Zeiger auf Schnittstellenzeiger. Wird gesetzt, wenn der verlangte Schnittstellentyp von der entsprechenden Instanz verfügbar ist.

Rückgabewert

Ein Rückgabewert S_OK weist auf Erfolg hin.

Wenn die verlangte Schnittstelle nicht verfügbar ist, gibt die Methode ADS_E_NOINTERFACE zurück.

Beschreibung

Abfrage der Referenz an eine implementierte Schnittstelle über der IID. Es wird empfohlen, Smart Pointer zu verwenden, um Schnittstellenzeiger zu initialisieren und zu halten.

Variante 1:

```
HRESULT GetTraceLevel(ITcUnkown* ip, TcTraceLevel& tl)
{
HRESULT hr = S_OK;
if (ip != NULL)
{
ITComObjectPtr spObj;
hr = ip->TcQueryInterface(spObj.GetIID(), &spObj);
if (SUCCEEDED(hr))
{
hr = spObj->TcGetObjPara(PID_TcTraceLevel, &tl, sizeof(tl));
}
return hr;
}
}
```

Die mit dem Smart Pointer verbundene Schnittstellen-ID kann in TcQueryInterface als Parameter verwendet werden. Der Operator „&“ wird den Zeiger auf die interne Schnittstellen-Zeiger-Membervariable des Smart Pointers zurückgeben. Variante 1 geht davon aus, dass der Schnittstellenzeiger initialisiert ist, wenn TcQueryInterface Erfolg anzeigt. Wenn der Bereich bleibt, dann gibt der Destructor des Smart Pointers spObj die Referenz frei.

Variante 2:

```
HRESULT GetTraceLevel(ITcUnkown* ip, TcTraceLevel& tl)
{
HRESULT hr = S_OK;
ITComObjectPtr spObj = ip;
if (spObj != NULL)
{
spObj->TcGetObjParam(PID_TcTraceLevel, &tl);
}
else
{
hr = ADS_E_NOINTERFACE;
}
return hr;
}
```

Wenn der Schnittstellenzeiger ip dem Smart Pointer spObj zugewiesen wird, dann wird die TcQueryInterface-Methode implizit aufgerufen mit IID_ITComObject auf der Instanz, auf die ip verweist. Dies führt zu einem kürzeren Code, aber der ursprüngliche Return-Code von TcQueryInterface geht verloren.

12.4.1.3 Methode ITcUnknown:TcRelease

Diese Methode dekrementiert den Referenzzähler.

Syntax

```
ULONG TcRelease( )
```

Rückgabewert

Daraus resultierender Referenzzählwert.

Beschreibung

Dekrementiert den Referenzzähler und gibt den neuen Wert zurück.

Wenn der Referenzzähler 0 wird, löscht das Objekt sich selber.

12.4.2 Schnittstelle ITComObject

Die ITComObject Schnittstelle wird von jedem TwinCAT Modul implementiert. Sie stellt grundlegende Funktionalitäten zur Verfügung.

Syntax

```
TCOM_DECL_INTERFACE("00000012-0000-0000-e000-000000000064", ITComObject)
struct __declspec(novtable) ITComObject: public ITcUnknown
```

Benötigtes include: `TcInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	TcGetObjectId(OTCID& objId) [▶ 124]	Speichert die Objekt ID mit Hilfe der gegebenen OTCID Referenz.
	OTCID TcGetObjectId [▶ 125]	Fragt die Objekt ID vom Objekt ab.
	TcSetObjectId [▶ 125]	Setzt die Objekt-ID des Objekts auf die gegebene OTCID
	TcGetObjectName [▶ 125]	Speichert den Objektnamen im Puffer mit der gegebenen Länge
	TcSetObjectName [▶ 126]	Setzt den Objektnamen des Objekts auf gegebenen CHAR*
	TcSetObjState [▶ 126]	Initialisiert einen Übergang zu einem vorgegebenen Zustand.
	TcGetObjState [▶ 127]	Fragt den aktuellen Zustands des Objekts ab.
	TcGetObjPara [▶ 127]	Fragt einen mit seiner PTCID identifizierten Objektparameter ab
	TcSetObjPara [▶ 128]	Setzt einen mit seiner PTCID identifizierten Objektparameter
	TcGetParentObjId [▶ 128]	Speichert die Parent-Objekt ID mit Hilfe der gegebenen OTCID Referenz.
	TcSetParentObjId [▶ 129]	Setzt die Parent-Objekt-ID auf die gegebene OTCID.
	TcTryToReleaseOpState [▶ 129]	Gibt Ressourcen frei; muss implementiert sein

Anmerkungen

Die ITComObject Schnittstelle wird von jedem TwinCAT Modul implementiert. Sie stellt Funktionalitäten zur Verfügung bezüglich der Zustandsmaschine und Informationen vom/an das TwinCAT System.

Sehen Sie dazu auch

- Schnittstelle ITcIoCyclic [▶ 146]

12.4.2.1 Methode ITcComObject:TcGetObjectId

Die Methode speichert Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

```
HRESULT TcGetObjectId( OTCID& objId )
```

Parameter

objId: (Typ: OTCID&) Referenz auf OTCID-Wert

Rückgabewert

Informiert über Erfolg der OTCID Abfrage.

Beschreibung

Die Methode speichert Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.2 Methode ITcComObject:TcGetObjectId

Die Methode speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

```
OTCID TcGetObjectId(void)
```

Parameter

Rückgabewert

OTCID: Gibt die OTCID des Objekts zurück.

Beschreibung

Die Methode speichert die Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.3 Methode ITcComObject:TcSetObjectId

Die Methode TcSetObjectId setzt die Objekt-ID des Objekts auf die gegebene OTCID.

Syntax

```
HRESULT TcSetObjectId( OTCID objId )
```

Parameter

objId: (Typ: OTCID) Die zu setzende OTCID.

Rückgabewert

Es wird empfohlen, stets S_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

Zeigt den Erfolg der id-Änderung an.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.4 Methode ITcComObject:TcGetObjectName

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

Syntax

```
HRESULT TcGetObjectName( CHAR* objName, ULONG nameLen );
```

Parameter

objName: (Typ: CHAR*) der zu setzende Name.

nameLen: (Typ: ULONG) die maximale, zu schreibende Länge.

Rückgabewert

Informiert über Erfolg der Namen-Abfrage.

Beschreibung

Die Methode TcGetObjectName speichert den Objektnamen im Puffer mit der gegebenen Länge.

Sehen Sie dazu auch

Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.5 Methode ITcComObject:TcSetObjectName

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts auf gegebenen CHAR*.

Syntax

```
HRESULT TcSetObjectName( CHAR* objName )
```

Parameter

objName: (Typ: CHAR*) der zu setzende Name des Objekts

Rückgabewert

Informiert über Erfolg der Namen-Abfrage.

Beschreibung

Die Methode TcSetObjectName setzt den Objekt-Namen des Objekts auf gegebenen CHAR*.

Sehen Sie dazu auch

Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.6 Methode ITcComObject:TcSetObjState

Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

Syntax

```
HRESULT TcSetObjState(TCOM_STATE state, ITComObjectServer* ipSrv, PTComInitDataHdr pInitData);
```

Parameter

state: (Typ: TCOM_STATE) stellt den neuen Zustand dar

ipSrv: (Typ: ITComObjectServer*) handhabt des Objekts

pInitData: (Typ: PTComInitDataHdr) zeigt auf eine Liste von Parametern (optional)
Siehe Makro IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA als Beispiel, wie die Liste iteriert werden kann.

Rückgabewert

Zeigt den Erfolg der Zustandsänderung an.

Beschreibung

Die Methode TcSetObjState initialisiert einen Übergang zum gegebenen Zustand.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.7 Methode ITcComObject:TcGetObjState

Die Methode TcGetObjState fragt den aktuellen Zustands des Objekts ab.

Syntax

```
HRESULT TcGetObjState(TCOM_STATE* pState)
```

Parameter

pState: (Typ: TCOM_STATE*) Zeiger auf den Zustand

Rückgabewert

Informiert über Erfolg der Zustandsabfrage.

Beschreibung

Die Methode TcGetObjState fragt den aktuellen Zustands des Objekts ab.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.8 Methode ITcComObject:TcGetObjPara

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

Syntax

```
HRESULT TcGetObjPara(PTCID pid, ULONG& nData, PVOID& pData, PTCGP pgp=0)
```

Parameter

pid: (Typ: PTCID) Parameter-ID des Objektparameters

nData: (Typ: ULONG&) max. Länge der Daten

pData: (Typ: PVOID&) Zeiger auf die Daten

pgp: (Typ: PTCGP) für zukünftige Erweiterung vorbehalten, NULL weitergeben

Rückgabewert

Informiert über Erfolg der Objektparameterabfrage.

Beschreibung

Die Methode TcGetObjPara fragt einen mittels seiner PTCID identifizierten Objektparameter ab.

Sehen Sie dazu auch

Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.9 Methode ITcComObject:TcSetObjPara

Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparameter.

Syntax

```
HRESULT TcSetObjPara(PTCID pid, ULONG nData, PVOID pData, PTCGP pgp=0)
```

Parameter

pid: (Typ: PTCID) Parameter-ID des Objektparameters

nData: (Typ: ULONG) max. Länge der Daten

pData: (Typ: PVOID) Zeiger auf die Daten

pgp: (Typ: PTCGP) für zukünftige Erweiterung vorbehalten, NULL weitergeben

Rückgabewert

Informiert über Erfolg der Objektparameterabfrage.

Beschreibung

Die Methode TcSetObjPara setzt einen mittels seiner PTCID identifizierten Objektparameter.

Sehen Sie dazu auch

Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.10 Methode ITcComObject:TcGetParentObjId

Die Methode TcGetParentObjId speichert Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

```
HRESULT TcGetParentObjId( OTCID& objId )
```

Parameter

objId: (Typ: OTCID&) Referenz auf OTCID-Wert

Rückgabewert

Informiert über Erfolg der parentObjId-Abfrage.

Beschreibung

Die Methode TcGetParentObjId speichert Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.11 Methode ITcComObject:TcSetParentObjId

Die Methode TcSetParentObjId setzt Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Syntax

```
HRESULT TcSetParentObjId( OTCID objId )
```

Parameter

objId: (Typ: OTCID) Referenz auf OTCID-Wert

Rückgabewert

Es wird empfohlen, stets S_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

Die Methode TcSetParentObjId setzt Parent-Objekt-ID mit Hilfe der gegebenen OTCID Referenz.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.2.12 Methode ITcComObject:TcTryToReleaseOpState

Die Methode TcTryToReleaseOpState gibt Ressourcen frei, z.B. Datenzeiger, um das Verlassen des OP-Zustands vorzubereiten.

Syntax

```
BOOL TcTryToReleaseOpState(void)
```

Parameter

Rückgabewert

TRUE bedeutet Freigabe von Ressourcen war erfolgreich.

Beschreibung

Die Methode TcTryToReleaseOpState gibt Ressourcen frei, z.B. Datenzeiger, um das Verlassen des OP-Zustands vorzubereiten. Muss implementiert werden, um mögliche wechselseitige Abhängigkeiten von Modulinstanzen aufzuheben. Siehe Beispiel10 [▶ 216].

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.3 Schnittstelle ITcCyclic

Die Schnittstelle ITcCyclic wird von TwinCAT Modulen implementiert, die ein Mal pro Task-Zyklus aufgerufen werden.

Syntax

```
TCOM_DECL_INTERFACE("03000010-0000-0000-e000-000000000064", ITcCyclic)
struct __declspec(novtable) ITcCyclic : public ITcUnknown
```

Benötigtes include: `TcIoInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	CycleUpdate [▶ 130]	Wird ein Mal pro Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen Aufrufer angemeldet ist.

Anmerkungen

Die Schnittstelle ITcCyclic wird von TwinCAT Modulen implementiert. Diese Schnittstelle wird der Methode ITcCyclicCaller::AddModule() übergeben, wenn sich ein Modul bei einem Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung wird die Methode CycleUpdate() der Modulinstanz aufgerufen.

12.4.3.1 Methode ITcCyclic:CyclicUpdate

Die Methode CyclicUpdate wird normalerweise von einem TwinCAT Task aufgerufen, nachdem die Schnittstelle angemeldet wurde.

Syntax

```
HRESULT TCOMAPI CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
```

Parameter

ipTask: (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

ipCaller: (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

Context: (Typ: ULONG_PTR) Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddModule() übergeben wurde.

Rückgabewert

Es wird empfohlen, stets S_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

In einem Taskzyklus wird die Methode CycleUpdate() aufgerufen, nachdem InputUpdate() für alle angemeldeten Modulinstanzen aufgerufen wurde. Demzufolge muss diese Methode verwendet werden, um eine zyklische Abarbeitung zu implementieren.

Sehen Sie dazu auch

- Schnittstelle ITcCyclicCaller [▶ 131]

12.4.4 Schnittstelle ITcCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.

Syntax

```
TCOM_DECL_INTERFACE("0300001E-0000-0000-e000-000000000064", ITcCyclicCaller)
struct __declspec(novtable) ITcCyclicCaller : public ITcUnknown
```

Benötigtes include: `TcIoInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	AddModule [▶ 131]	Modul anmelden, das die ITcCyclic Schnittstelle implementiert.
	RemoveModule [▶ 132]	Die zuvor angemeldete ITcCyclic Schnittstelle eines Moduls abmelden.

Anmerkungen

Die ITcCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle, um seine ITcCyclic Schnittstelle bei einem Task anzumelden, normalerweise als letzten Initialisierungsschritt beim Übergang SafeOP zu OP. Nach der Anmeldung wird die Methode CycleUpdate() der Modulinstanz aufgerufen. Die Schnittstelle wird ebenfalls zum Abmelden des Moduls verwendet, damit es nicht mehr vom Task aufgerufen wird.

12.4.4.1 Methode ITcCyclicCaller:AddModule

Meldet die ITcCyclic Schnittstelle eines Moduls bei einem zyklischen Aufrufer, z.B. einem TwinCAT Task an.

Syntax

```
virtual HRESULT TCOMAPI
AddModule(STcCyclicEntry* pEntry, ITcCyclic* ipMod, ULONG_PTR
context=0, ULONG sortOrder=0)=0;
```

Parameter

pEntry: (Typ: STcCyclicEntry) [in] Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen Aufrufers eingefügt wird. Siehe auch Beschreibung.

ipMod: (Typ: ITcCyclic) [in] Schnittstellenzeiger, der vom zyklischen Aufrufer verwendet wird

context: (Typ: ULONG_PTR) [optional] ein Kontextwert, der bei jedem Aufruf an die ITcCyclic::CyclicUpdate() Methode übergeben wird.

sortOrder: (Typ: ULONG) [optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

Rückgabewert

Typ: HRESULT

Bei Erfolg gibt die Methode S_OK zurück. Wenn der zyklische Aufrufer, d.h. der TwinCAT Task, nicht im OP-Zustand ist, wird der Fehler ADS_E_INVALIDSTATE zurückgegeben.

Beschreibung

Eine TwinCAT-Modulklasse verweist normalerweise mit einem Smart Pointer auf den zyklischen Aufrufer Typ ITcCyclicCallerPtr. Die Objekt-ID des Tasks ist in diesem Smart Pointer gespeichert und eine Referenz zum Task kann über den TwinCAT Objektserver erhalten werden. Darüber hinaus enthält die Klasse des Smart Pointers bereits einen Listeneintrag. Demzufolge kann der Smart Pointer als erster Parameter für die AddModule Methode verwendet werden.

Der folgende Beispielcode zeigt die Anmeldung der ITcCyclicCaller Schnittstelle.

```
RESULT hr =
S_OK;

if ( m_spCyclicCaller.HasOID() ) {

if ( SUCCEEDED_DBG(hr =
m_spSrv->TcQuerySmartObjectInterface(m_spCyclicCaller)) )
{

    if ( FAILED(hr =
m_spCyclicCaller->AddModule(m_spCyclicCaller,
THIS_CAST(ITcCyclic))) ) {

     m_spCyclicCaller = NULL;

    }

}

}
```

Sehen Sie dazu auch

- Schnittstelle ITcCyclic [▶ 130]
- Schnittstelle ITcIoCyclicCaller [▶ 147]

12.4.4.2 Methode ITcCyclicCaller:RemoveModule

Eine Modulinstanz vom Aufruf durch einen zyklischen Aufrufer abmelden.

Syntax

```
virtual HRESULT TCOMAPI
RemoveModule(STcCyclicEntry* pEntry)=0;
```

Parameter

pEntry: (Typ: STcCyclicEntry) verweist auf den Listeneintrag, der aus der internen Liste des zyklischen Aufrufers zu entfernen ist.

Rückgabewert

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E_FAIL zurück.

Beschreibung

Vergleichbar mit der Methode AddModule() wird der Smart Pointer für den zyklischen Aufrufer als Listeneintrag verwendet, wenn die Modulinstanz aus dem zyklischen Aufrufer entfernt werden soll.

Deklaration und Verwendung des Smart Pointers:

ITcCyclicCallerInfoPtr m_spCyclicCaller;

```
if (
m_spCyclicCaller ) {
m_spCyclicCaller->RemoveModule(m_spCyclicCaller);
}
m_spCyclicCaller = NULL;
```

12.4.5 Schnittstelle ITcFileAccess

Schnittstelle für Zugriff auf Dateisystem von TwinCAT C++ Modulen aus

Syntax

```
TCOM_DECL_INTERFACE("742A7429-DA6D-4C1D-80D8-398D8C1F1747", ITcFileAccess) __declspec(novtable) ITc-
FileAccess: public ITcUnknown
```

Benötigtes include: `TcFileAccessInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	FileOpen [▶ 133]	Öffnet eine Datei
	FileClose [▶ 134]	Schließt eine Datei
	FileRead [▶ 135]	Liest aus einer Datei
	FileWrite [▶ 135]	Schreibt in eine Datei
	FileSeek [▶ 136]	Setzt Position in einer Datei
	FileTell [▶ 136]	Fragt Position in einer Datei ab
	FileRename [▶ 137]	Nennt eine Datei um
	FileDelete [▶ 137]	Löscht eine Datei
	FileGetStatus [▶ 138]	Erhält den Zustand einer Datei
	FileFindFirst [▶ 138]	Sucht nach einer Datei, erste Iteration
	FileFindNext [▶ 139]	Sucht nach einer Datei, nächste Iteration
	FileFindClose [▶ 140]	Schließt eine Dateisuche
	MkDir [▶ 140]	Erstellt ein Verzeichnis
	RmDir [▶ 141]	Löscht ein Verzeichnis

Anmerkungen

Die ITcFileAccess Schnittstelle wird für den Zugriff auf Dateien in Dateisystemen verwendet.
Weil die zur Verfügung gestellten Methoden Blockaden verursachen, sollte diese Schnittstelle nicht in CycleUpdate() / Echtzeitkontext verwendet werden. Die abgeleitete Schnittstelle ITcFileAccessAsync [▶ 141] fügt eine Check() Methode hinzu, die stattdessen verwendet werden kann.

Siehe Beispiel20a: FileIO-Cyclic Read / Write [▶ 250].

Die Schnittstelle wird über die Modulklasse CID_TcFileAccess implementiert.

12.4.5.1 Methode ITcFileAccess:FileOpen

Öffnet eine Datei

Syntax

```
virtual HRESULT TCOMAPI FileOpen(PCCH szFileName, TcFileAccessMode AccessMode, PTcFileHandle phFi-
le);
```

Parameter

szFileName: (Typ: PCCH) [in] der Name der zu öffnenden Datei

AccessMode: (Typ: TcFileAccessMode) [in] Art des Zugriffs auf die Datei, siehe *TcFileAccessServices.h*

phFile: (Typ: TcFileHandle) [out] zurückgegebener Datei-Handle

Rückgabewert

Typ: HRESULT

Bei Erfolg gibt die Methode S_OK zurück.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Die Methode gibt einen Handle für den Zugriff auf die Datei zurück, dessen Name in szFileName definiert ist.

AccessModes können folgendermaßen verwendet werden:

```
typedef enum TcFileAccessMode
{
amRead = 0x00000001,
amWrite = 0x00000002,
amAppend = 0x00000004,
amPlus = 0x00000008,
amBinary = 0x00000010,
amReadBinary = 0x00000011,
amWriteBinary = 0x00000012,
amText = 0x00000020,
amReadText = 0x00000021,
amWriteText = 0x00000022,
amEnsureDirectory = 0x00000040,
amReadBinaryED = 0x00000051,
amWriteBinaryED = 0x00000052,
amReadTextED = 0x00000061,
amWriteTextED = 0x00000062,
amEncryption = 0x00000080,
amReadBinEnc = 0x00000091,
amWriteBinEnc = 0x00000092,
amReadBinEncED = 0x000000d1,
amWriteBinEncED = 0x000000d2,
} TcFileAccessMode, *PTcFileAccessMode;
```

12.4.5.2 Methode ITcFileAccess:FileClose

Schließt eine Datei

Syntax

```
virtual HRESULT TCOMAPI FileClose(PTcFileHandle phFile);
```

Parameter

phFile: (Typ: TcFileHandle) [out] zurückgegebener Datei-Handle

Rückgabewert

Typ: HRESULT

Bei Erfolg gibt die Methode S_OK zurück.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Die Methode schließt eine mittels phFile definierte Datei.

Sehen Sie dazu auch

ADS Return Codes [▶ 260]

12.4.5.3 Methode ITcFileAccess:FileRead

Liest Daten aus einer Datei.

Syntax

```
virtual HRESULT TCOMAPI
FileRead(TcFileHandle hFile, PVOID pData, UINT cbData, PUINT pcbRead);
```

Parameter

hFile: (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

pData: (Typ: PVOID) [out] Speicherort der zu lesenden Daten

cbData: (Typ: PVOID) [in] maximale Größe der zu lesenden Daten (Größe des Speichers hinter pData)

pcbRead: (Typ: PUINT) [out] Größe der gelesenen Daten

Rückgabewert

Typ: HRESULT

Wenn Daten gelesen werden können, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode fragt Daten von einer mit Dateihandle definierten Datei ab. Die Daten werden in pData gespeichert, wobei pcbRead die Länge der Daten angibt.

12.4.5.4 Methode ITcFileAccess:FileWrite

Daten in eine Datei schreiben.

Syntax

```
virtual HRESULT TCOMAPI
FileWrite(TcFileHandle hFile, PCVOID pData, UINT cbData, PUINT pcbWrite);
```

Parameter

hFile: (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

pData: (Typ: PVOID) [in] Speicherort der zu schreibenden Daten

cbData: (Typ: PVOID) [in] Größe der zu schreibenden Daten (Größe des Speichers hinter pData)

pcbRead: (Typ: PUINT) [out] Größe der geschriebenen Daten

Rückgabewert

Typ: HRESULT

Wenn Daten geschrieben werden können, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode schreibt Daten in eine mit Dateihandle definierte Datei. Die Daten werden in pData gelesen, wobei pcbRead die Länge der Daten angibt.

12.4.5.5 Methode ITcFileAccess:FileSeek

Setzt Position in einer Datei.

Syntax

```
virtual HRESULT TCOMAPI FileSeek(TcFileHandle hFile, UINT uiPos);
```

Parameter

hFile: (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

uiPos: (Typ: UINT) [in] Position auf die zu setzen ist

Rückgabewert

Typ: HRESULT

Wenn die Position gesetzt werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode legt die Position innerhalb einer Datei für weitere Aktionen fest.

12.4.5.6 Methode ITcFileAccess:FileTell

Fragt Position in einer Datei ab.

Syntax

```
virtual HRESULT TCOMAPI FileTell(TcFileHandle hFile, PUINT puiPos);
```

Parameter

hFile: (Typ: TcFileHandle) [in] verweist auf die zuvor geöffnete Datei

puiPos: (Typ: PUINT) [out] Speicherort der zurückzugebenden Position

Rückgabewert

Typ: HRESULT

Wenn die Position abgefragt werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode fragt die derzeit gesetzte Position innerhalb einer Datei ab.

12.4.5.7 Methode ITcFileAccess:FileRename

Nennt eine Datei um

Syntax

```
virtual HRESULT TCOMAPI FileRename(PCCH szOldName, PCCH szNewName);
```

Parameter

szOldName: (Typ: PCCH) [in] der umzubenennende Dateiname

szNewName: (Typ: PCCH) [in] der neue Dateiname

Rückgabewert

Typ: HRESULT

Wenn die Datei umbenannt werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode benennt eine Datei mit einem neuen Namen um.

12.4.5.8 Methode ITcFileAccess:FileDelete

Nennt eine Datei um

Syntax

```
virtual HRESULT TCOMAPI FileDelete(PCCH szFileName);
```

Parameter

szFileName: (Typ: PCCH) [in] Name der zu löschende Datei

Rückgabewert

Typ: HRESULT

Wenn die Datei gelöscht werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode löscht eine Datei aus dem Dateisystem

12.4.5.9 Methode ITcFileAccess:FileGetStatus

Fragt den Zustand einer Datei ab.

Syntax

```
virtual HRESULT TCOMAPI FileGetStatus(PCCH szFileName, PTcFileStatus pFileStatus));
```

Parameter

szFileName: (Typ: PCCH) [in] der Name der fraglichen Datei

pFileStatus: (Typ: PTcFileStatus) [out] der Zustand der Datei. Vergleiche mit *TcFileAccessServices.h*

Rückgabewert

Typ: HRESULT

Wenn Zustand zurückgegeben werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode fragt Zustandsinformationen bezüglich eines gegebenen Dateinamen ab.

Dazu gehören die folgenden Informationen:

```
typedef struct TcFileStatus
{
union
{
ULONGLONG ulFileSize;
struct
{
ULONG ulFileSizeLow;
ULONG ulFileSizeHigh;
};
};
ULONGLONG ulCreateTime;
ULONGLONG ulModifiedTime;
ULONGLONG ulReadTime;
DWORD dwAttribute;
DWORD wReserved0;
} TcFileStatus, *PTcFileStatus;
```

12.4.5.10 Methode ITcFileAccess:FileFindFirst

Möglichkeit, die Dateien eines Verzeichnisses zu durchlaufen.

Syntax

```
virtual HRESULT TCOMAPI FileFindFirst (PCCH szFileName, PTcFileFindData pFileFindData , PTcFileFind-
Handle phFileFind);
```

Parameter

szFileName: (Typ: PCCH) [in] Verzeichnis oder Pfad, und Name der gesuchten Datei. Der Dateiname kann Platzhalterzeichen wie Sternchen (*) oder Fragezeichen (?) enthalten.

pFileFindData: (Typ: PTcFileFindData) [out] die Beschreibung der ersten Datei. Vergleiche mit *TcFileAccessServices.h*

phFileFind: (Typ: PTcFileFindHandle) [out] Handle um weiter mit FileFindNext zu suchen.

Rückgabewert

Typ: HRESULT

Wenn eine Datei gefunden werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode beginnt mit der Suche nach Dateien in einem vorgegebenen Verzeichnis. Die Methode gewährt Zugriff auf PTcFileFindData der ersten gefundenen Datei, mit folgenden Informationen:

```
typedef struct TcFileFindData
{
TcFileHandle hFile;
DWORD dwFileAttributes;
ULONGLONG ui64CreationTime;
ULONGLONG ui64LastAccessTime;
ULONGLONG ui64LastWriteTime;
DWORD dwFileSizeHigh;
DWORD dwFileSizeLow;
DWORD dwReserved1;
DWORD dwReserved2;
CHAR cFileName[260];
CHAR cAlternateFileName[14];
WORD wReserved0;
} TcFileFindData, *PTcFileFindData;
```

12.4.5.11 Methode ITcFileAccess:FileFindNext

Die Dateien eines Verzeichnisses weiter durchlaufen.

Syntax

```
virtual HRESULT TCOMAPI FileFindNext (TcFileFindHandle hFileFind, PTcFileFindData pFileFindData);
```

Parameter

hFileFind: (Typ: PTcFileFindHandle) [in] Handle um weiter mit FileFindNext zu suchen.

pFileFindData: (Typ: PTcFileFindData) [out] die Beschreibung der nächsten Datei. Vergleiche mit *TcFileAccessServices.h*

Rückgabewert

Typ: HRESULT

Wenn eine Datei gefunden werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode sucht nach der nächsten Datei in einem Verzeichnis. Die Methode gewährt Zugriff auf PTcFileFindData der gefundenen Datei, mit folgenden Informationen:

```
typedef struct TcFileFindData
{
TcFileHandle hFile;
DWORD dwFileAttributes;
ULONGLONG ui64CreationTime;
ULONGLONG ui64LastAccessTime;
ULONGLONG ui64LastWriteTime;
DWORD dwFileSizeHigh;
DWORD dwFileSizeLow;
DWORD dwReserved1;
DWORD dwReserved2;
CHAR cFileName[260];
CHAR cAlternateFileName[14];
WORD wReserved0;
} TcFileFindData, *PTcFileFindData;
```

12.4.5.12 Methode ITcFileAccess:FileFindClose

Die Suche nach Dateien in einem Verzeichnis schließen.

Syntax

```
virtual HRESULT TCOMAPI FileFindClose (TcFileFindHandle hFileFind);
```

Parameter

hFileFind: (Typ: PTcFileFindHandle) [in] Handle um die Suche zu schließen

Rückgabewert

Typ: HRESULT

Wenn die Dateisuche geschlossen werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode schließt die Suche nach Dateien in einem Verzeichnis.

12.4.5.13 Methode ITcFileAccess:MkDir

Ein Verzeichnis in einem Dateisystem erstellen.

Syntax

```
virtual HRESULT TCOMAPI MkDir(PCCH szDir);
```

Parameter

szDir: (Typ: PCCH) [in] zu erstellendes Verzeichnis

Rückgabewert

Typ: HRESULT

Wenn Verzeichnis erstellt werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode erstellt ein Verzeichnis entsprechend der Definition im szDir Parameter.

12.4.5.14 Methode ITcFileAccess:RmDir

Ein Verzeichnis aus dem Dateisystem entfernen.

Syntax

```
virtual HRESULT TCOMAPI RmDir(PCCH szDir);
```

Parameter

szDir: (Typ: PCCH) [in] zu löschendes Verzeichnis

Rückgabewert

Typ: HRESULT

Wenn Verzeichnis gelöscht werden konnte, wird S_OK zurückgegeben.

Besonders interessante Fehlercodes:

- ADS_E_TIMEOUT wenn Zeitüberschreitung (5 Sekunden) abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Methode löscht ein Verzeichnis entsprechend der Definition im szDir Parameter.

12.4.6 Schnittstelle ITcFileAccessAsync

Asynchroner Zugriff auf Dateioperationen.
Diese Schnittstelle erweitert ITcFileAccess [▶ 133].

Syntax

```
TCOM_DECL_INTERFACE("C04AC244-C126-466E-982E-93EC571F2277", ITcFileAccessAsync) struct __decl-
spec(novtable) ITcFileAccessAsync: public ITcFileAccess
```

Benötigtes include: `TcFileAccessInterfaces.h`

Methoden

Symbol	Name	Beschreibung
	Überprüfen [▶ 142]	Zustand der Dateioperation abfragen

Schnittstellenparameter

Sym-bol	Name	Beschreibung
	PID_TcFileAccessAsyncSegmentSize	Größe der an System-Service übergebenen Segmente
	PID_TcFileAccessAsyncTimeoutMs	Schließt eine Datei
	PID_TcFileAccessAsyncNetId(Str)	NetId des zu kontaktierenden System-Service

Anmerkungen

Schnittstelle kann von Modulinstanz mit Klassen-ID CID_TcFileAccessAsync erhalten werden.
Bei Verwendung der asynchronen Schnittstelle, geben die von der synchronen Variante geerbten Schnittstellenmethoden ADS_E_PENDING zurück, wenn eine Abfrage erfolgreich unterbreitet, aber noch nicht abgeschlossen wurde. Wenn der Aufruf eingeht, während die vorherige Anfrage immer noch abgearbeitet wurde, wird der Fehlercode ADS_E_BUSY zurückgegeben.

Beschreibung der Modulparameter:

- PID_TcFileAccessAsyncAdsProvider: Objekt-ID eines Tasks, das die ADS-Schnittstelle bereitstellt.
- PID_TcFileAccessAsyncNetId / PID_TcFileAccessAsyncNetIdStr: AmsNetId des System-Service, der für den Dateizugriff verwendet wird. Die „Str“ Variante nimmt die AmsNetId als Zeichenkette. Bitte eins verwenden.
- PID_TcFileAccessAsyncTimeoutMs: Zeitüberschreitung für einen Dateizugriff
- PID_TcFileAccessAsyncSegmentSize: Der Lese- und Schreibzugriff auf Datei wird mit dieser Segmentgröße fragmentiert.

Siehe Beispiel20a: FileIO-Cyclic Read / Write [▶ 250]

Sehen Sie dazu auch

- Method ITcFileAccessAsync::Check() [▶ 142]
- Beispiel20a: FileIO-Cyclic Read / Write [▶ 250]

12.4.6.1 Method ITcFileAccessAsync::Check()

Zustand der Dateioperation abfragen

Syntax

```
virtual HRESULT TCOMAPI Check();
```

Parameter

keiner

Rückgabewert

Typ: HRESULT

Gibt S_OK zurück, wenn Dateioperation abgeschlossen ist

Besonders interessante Fehlercodes:

- ADS_E_PENDING, wenn die Dateioperation nicht abgeschlossen ist.
- ADS_E_TIMEOUT, wenn die Zeitüberschreitung für die Dateioperation abgelaufen ist.

Weitere ADS-Zustandscodes [▶ 260] sind möglich.

Beschreibung

Diese Operation überprüft den Zustand der zuvor aufgerufenen Dateioperation

12.4.7 Schnittstelle ITcPostCyclic

Die Schnittstelle wird von TwinCAT Modulen implementiert, die ein Mal pro Taskzyklus im Anschluss an die Ausgang-Aktualisierung aufgerufen werden (vergleichbar mit Attribut TcCallAfterOutputUpdate der SPS).

Syntax

```
TCOM_DECL_INTERFACE("03000025-0000-0000-e000-000000000064", ITcPostCyclic)
struct __declspec(novtable) ITcPostCyclic : public ITcUnknown
```

Benötigtes include: `TcIoInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	PostCycleUpdate [▶ 143]	Wird ein Mal pro Taskzyklus nach der Ausgangs-Aktualisierung aufgerufen, wenn die Schnittstelle bei einem zyklischen Aufrufer angemeldet wurde.

Anmerkungen

Die ITcPostCyclic Schnittstelle wird von TwinCAT Modulen implementiert. Diese Schnittstelle wird der Methode ITcCyclicCaller::AddPostModule() übergeben, wenn ein Modul sich selber bei einem Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung wird die Methode PostCycleUpdate() der Modulinstanz aufgerufen.

12.4.7.1 Methode ITcPostCyclic:PostCyclicUpdate

Die normalerweise von einem TwinCAT Task nach der Ausgang-Aktualisierung aufgerufene Methode PostCyclicUpdate, nachdem die Schnittstelle angemeldet wurde.

Syntax

```
HRESULT TCOMAPI PostCycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
```

Parameter

ipTask: (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

ipCaller: (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

Context: (Typ: ULONG_PTR) Kontext beinhaltet den Wert, der an die Methode ITcPostCyclicCaller::AddPostModule() übergeben wurde.

Rückgabewert

Es wird empfohlen, stets S_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

Innerhalb eines Taskzyklusses wird die Methode PostCycleUpdate() aufgerufen, nachdem OutputUpdate() für alle angemeldeten Modulinstanzen aufgerufen wurde. Demzufolge muss diese Methode verwendet werden, um derlei zyklische Abarbeitung zu implementieren.

Sehen Sie dazu auch

- Schnittstelle ITcPostCyclicCaller [▶ 144]

12.4.8 Schnittstelle ITcPostCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcPostCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.

Syntax

```
TCOM_DECL_INTERFACE("03000026-0000-0000-e000-000000000064", ITcCyclicCaller)
struct __declspec(novtable) ITcPostCyclicCaller : public ITcUnknown Ca
```

Benötigtes include: `TcIoInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	AddPostModule [▶ 144]	Modul anmelden, das die ITcPostCyclic Schnittstelle implementiert.
	RemovePostModule [▶ 145]	Zuvor angemeldete ITcPostCyclic Schnittstelle eines Moduls abmelden.

Anmerkungen

Die ITcPostCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle um seine ITcPostCyclic Schnittstelle bei einem Task anzumelden, normalerweise als letzten Initialisierungsschritt beim SafeOP zum OP Übergang. Nach der Anmeldung wird die Methode PostCycleUpdate() der Modulinstanz aufgerufen. Die Schnittstelle wird ebenfalls zum Abmelden des Moduls verwendet, damit es nicht mehr vom Task aufgerufen wird.

12.4.8.1 Methode ITcPostCyclicCaller:AddPostModule

Meldet die ITcPostCyclic Schnittstelle eines Moduls bei einem zyklischen Aufrufer, z.B. einem TwinCAT Task an.

Syntax

```
virtual HRESULT TCOMAPI
AddPostModule(STcPostCyclicEntry* pEntry, ITcPostCyclic* ipMod, ULONG_PTR
context=0, ULONG sortOrder=0)=0;
```

Parameter

pEntry: (Typ: STcPostCyclicEntry) [in] Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen Aufrufers eingefügt wird. Siehe auch Beschreibung.

ipMod: (Typ: ITcPostCyclic) [in] Schnittstellenzeiger, der vom zyklischen Aufrufer verwendet wird

context: (Typ: ULONG_PTR) [optional] ein Kontextwert, der bei jedem Aufruf an die ITcPostCyclic::PostCyclicUpdate() Methode übergeben wird.

sortOrder: (Typ: ULONG) [optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

Rückgabewert

Typ: HRESULT

Bei Erfolg gibt die Methode S_OK zurück. Wenn der zyklische Aufrufer, d.h. der TwinCAT Task, nicht im OP-Zustand ist, wird der Fehler ADS_E_INVALIDSTATE zurückgegeben.

Beschreibung

Ein TwinCAT-Modulklasse verwendet einen Smart Pointer, um auf den zyklischen Aufrufer vom Typ ITcPostCyclicCallerPtr zu verweisen. Die Objekt-ID des Tasks ist in diesem Smart Pointer gespeichert und eine Referenz zum Task kann über den TwinCAT Objektserver erhalten werden. Darüber hinaus enthält die Klasse des Smart Pointers bereits einen Listeneintrag. Demzufolge kann der Smart Pointer als erster Parameter für die AddPostModule Methode verwendet werden.

Das folgende Codebeispiel veranschaulicht die Anmeldung der ITcPostCyclicCaller Schnittstelle.

```
RESULT hr =
S_OK;

if ( m_spPostCyclicCaller.HasOID() ) {

if ( SUCCEEDED_DBG(hr =
m_spSrv->TcQuerySmartObjectInterface(m_spPostCyclicCaller)) )
{

    if ( FAILED(hr =
m_spPostCyclicCaller->AddPostModule(m_spPostCyclicCaller,
THIS_CAST(ITcPostCyclic))) ) {

     m_spPostCyclicCaller = NULL;

    }

}

}
```

Sehen Sie dazu auch

- Schnittstelle ITcPostCyclic [▶ 143]
- Schnittstelle ITcIoCyclicCaller [▶ 147]

12.4.8.2 Methode ITcPostCyclicCaller:RemovePostModule

Eine Modulinstanz vom Aufruf durch einen zyklischen Aufrufer abmelden.

Syntax

```
virtual HRESULT TCOMAPI
RemovePostModule(STcPostCyclicEntry* pEntry)=0;
```

Parameter

pEntry: (Typ: STcPostCyclicEntry) verweist auf den Listeneintrag, der aus der internen Liste des zyklischen Aufrufers zu entfernen ist.

Rückgabewert

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E_FAIL zurück.

Beschreibung

Vergleichbar mit der Methode AddPostModule() wird der Smart Pointer für den zyklischen Aufrufer als Listeneintrag verwendet, wenn die Modulinstanz aus dem zyklischen Aufrufer entfernt werden soll.

Deklaration und Verwendung des Smart Pointers:

ITcPostCyclicCallerInfoPtr m_spPostCyclicCaller;

```
if (
m_spPostCyclicCaller ) {
m_spPostCyclicCaller->RemovePostModule(m_spPostCyclicCaller);

}

m_spPostCyclicCaller = NULL;
```

12.4.9 Schnittstelle ITcIoCyclic

Diese Schnittstelle wird von TwinCAT Modulen implementiert, die bei Eingang-Aktualisierung und bei Ausgang-Aktualisierung innerhalb eines Taskzyklus aufgerufen werden.

Syntax

```
TCOM_DECL_INTERFACE("03000011-0000-0000-e000-000000000064", ITcIoCyclic)
struct __declspec(novtable) ITcIoCyclic : public ITcUnknown
```

Benötigtes include: `TcIoInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	InputUpdate [▶ 146]	Wird zu Beginn eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.
	OutputUpdate [▶ 147]	Wird am Ende eines Taskzyklus aufgerufen, wenn die Schnittstelle bei einem zyklischen I/O-Aufrufer angemeldet ist.

Anmerkungen

ITcIoCyclic kann dazu verwendet werden, ein TwinCAT Modul zu implementieren, das als Feldbustreiber oder I/O-Filtermodul agiert.

Diese Schnittstelle wird der Methode ITcIoCyclicCaller::AddIoDriver übergeben, wenn sich ein Modul bei einem Task anmeldet, normalerweise als letzter Initialisierungsschritt beim Übergang von SafeOP zu OP. Nach der Anmeldung werden die Methoden InputUpdate() und OutputUpdate() der Modulinstanz je einmal pro Taskzyklus aufgerufen.

12.4.9.1 Methode ITcIoCyclic:InputUpdate

Die Methode InputUpdate wird normalerweise von einem TwinCAT Task aufgerufen, nachdem die Schnittstelle angemeldet wurde.

Syntax

```
virtual HRESULT TCOMAPI InputUpdate(ITcTask* ipTask,
ITcUnknown* ipCaller, DWORD dwStateIn, ULONG_PTR context = 0)=0;
```

Parameter

ipTask: (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

ipCaller: (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

dwStateIn: (Typ: DWORD) zukünftigen Erweiterungen vorbehalten, derzeit ist dieser Wert immer 0
context: (Typ: ULONG_PTR) Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddIoDriver() übergeben wurde.

Rückgabewert

Es wird empfohlen, stets S_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

In einem Taskzyklus wird die Methode InputUpdate() für alle angemeldeten Modulinstanzen zuerst aufgerufen. Deswegen muss diese Methode für die Aktualisierung der Datenbereiche vom Typ Eingabequelle des Moduls verwendet werden.

12.4.9.2 Methode ITcIoCyclic:OutputUpdate

Die Methode OutputUpdate wird normalerweise von einem TwinCAT Task aufgerufen, nachdem die Schnittstelle angemeldet wurde.

Syntax

```
virtual HRESULT TCOMAPI OutputUpdate(ITcTask* ipTask, ITcUnknown* ipCaller,
PDWORD pdwStateOut = NULL, ULONG_PTR context = 0)=0;
```

Parameter

ipTask: (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

ipCaller: (Typ: ITcUnknown) verweist auf die aufrufende Instanz.

pdwStateOut: (Typ: DWORD) [out] zukünftigen Erweiterungen vorbehalten, derzeit wird der Rückgabewert ignoriert.
context: (Typ: ULONG_PTR) Kontext beinhaltet den Wert, der an die Methode ITcCyclicCaller::AddIoDriver() übergeben wurde.

Rückgabewert

Es wird empfohlen, stets S_OK zurückzugeben. Derzeit wird der Rückgabewert von den TwinCAT Tasks ignoriert.

Beschreibung

In einem Taskzyklus wird für alle angemeldeten Modulinstanzen die Methode OutputUpdate() aufgerufen. Deswegen muss diese Methode für die Aktualisierung der Datenbereiche vom Typ Eingabequelle des Moduls verwendet werden.

12.4.10 Schnittstelle ITcIoCyclicCaller

Schnittstelle zum Anmelden oder Abmelden der ITcIoCyclic Schnittstelle eines Moduls bei einem TwinCAT Task.

Syntax

```
TCOM_DECL_INTERFACE("0300001F-0000-0000-e000-000000000064", ITcIoCyclicCaller)
struct __declspec(novtable) ITcIoCyclicCaller : public ITcUnknown
```

Benötigtes include: `TcIoInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	AddIoDriver [▶ 148]	Modul anmelden, das die ITcIoCyclic Schnittstelle implementiert.
	RemoveIoDriver [▶ 149]	Zuvor angemeldete ITcIoCyclic Schnittstelle eines Moduls abmelden.

Anmerkungen

Die ITcIoCyclicCaller Schnittstelle wird von TwinCAT Tasks implementiert. Ein Modul verwendet diese Schnittstelle um seine ITcIoCyclic Schnittstelle bei einem Task anzumelden, normalerweise als letzten Initialisierungsschritt beim SafeOP zum OP Übergang. Nach der Anmeldung wird die Methode CycleUpdate() der Modulinstanz aufgerufen. Die Schnittstelle wird ebenfalls zum Abmelden des Moduls verwendet, damit es nicht mehr vom Task aufgerufen wird.

12.4.10.1 Methode ITcIoCyclicCaller:AddIoDriver

Meldet die ITcIoCyclic Schnittstelle eines Moduls bei einem zyklischen I/O-Aufrufer, z.B. einem TwinCAT Task an.

Syntax

```
virtual HRESULT TCOMAPI AddIoDriver(STcIoCyclicEntry*
pEntry, ITcIoCyclic* ipDrv, ULONG_PTR context=0, ULONG sortOrder=0)=0;
```

Parameter

pEntry: (Typ: STcIoCyclicEntry) Zeiger auf einen Listeneintrag, welcher in die interne Liste des zyklischen I/O-Aufrufers eingefügt wird. Siehe auch Beschreibung.

ipDrv: (Typ: ITcIoCyclic) [in] Schnittstellenzeiger, der vom zyklischen I/O-Aufrufer verwendet wird

context: (Typ: ULONG_PTR) [optional] ein Kontextwert, der bei jedem Aufruf an die ITcIoCyclic::InputUpdate() und ITcIoCyclic::OutputUpdate Methode übergeben wird.

sortOrder: (Typ: ULONG) [optional] die Sortierreihenfolge kann für die Steuerung der Ausführungsreihenfolge verwendet werden, wenn verschiedene Modulinstanzen vom gleichen zyklischen Aufrufer ausgeführt werden.

Rückgabewert

Typ: HRESULT

Beschreibung

Eine TwinCAT-Modulklasse verwendet normalerweise einen Smart Pointer, um auf den zyklischen I/O-Aufrufer vom Typ ITcIoCyclicCallerPtr zu verweisen. Die Objekt-ID des zyklischen I/O-Aufrufers ist in diesem Smart Pointer gespeichert und eine Referenz kann über den TwinCAT Objektserver erhalten werden. Darüber hinaus enthält die Klasse des Smart Pointers bereits einen Listeneintrag. Demzufolge kann der Smart Pointer als erster Parameter für die AddIoDriver Methode verwendet werden.

Das folgende Codebeispiel veranschaulicht die Anmeldung der ITcIoCyclicCaller Schnittstelle.

```
HRESULT hr = S_OK;
if ( m_spIoCyclicCaller.HasOID() )
{
if ( SUCCEEDED_DBG(hr = m_spSrv->TcQuerySmartObjectInterface(m_spIoCyclicCaller))
)
{
if ( FAILED(hr = m_spIoCyclicCaller->AddIoDriver(m_spIoCyclicCaller,
THIS_CAST(ITcIoCyclic))) )
{
m_spIoCyclicCaller = NULL;
    }
}
}
```

12.4.10.2 Methode ITcIoCyclicCaller:RemoveIoDriver

Eine Modulinstanz vom Aufruf durch einen zyklischen I/O-Aufrufer abmelden.

Syntax

```
virtual HRESULT TCOMAPI
RemoveIoDriver(STcIoCyclicEntry* pEntry)=0;
```

Parameter

pEntry: (Typ: STcIoCyclicEntry) verweist auf den Listeneintrag, der aus der internen Liste des zyklischen I/O-Aufrufers zu entfernen ist.

Rückgabewert

Wenn der Eintrag nicht in der internen Liste ist, gibt die Methode E_FAIL zurück.

Beschreibung

Vergleichbar mit der Methode AddIoDriver() wird der Smart Pointer für den zyklischen I/O-Aufrufer als Listeneintrag verwendet, wenn die Modulinstanz aus dem zyklischen I/O-Aufrufer entfernt werden soll.

Deklaration und Verwendung des Smart Pointers:

```
ITcIoCyclicCallerInfoPtr
m_spIoCyclicCaller;
if ( m_spIoCyclicCaller )
{
m_spIoCyclicCaller->RemoveIoDriver(m_spIoCyclicCaller);
}
m_spCyclicCaller = NULL;
```

12.4.11 Schnittstelle ITcRTimeTask

Abfrage von erweiterten TwinCAT Taskinformationen.

Syntax

```
TCOM_DECL_INTERFACE("02000003-0000-0000-e000-000000000064", ITcRTimeTask)
struct __declspec(novtable) ITcRTimeTask : public ITcTask
```

Benötigtes include: `TcRtInterfaces.h`

Methoden

Symbol	Name	Beschreibung
	GetCpuAccount [▶ 150]	Abfrage des CPU-Kontos eines TwinCAT Tasks.

Anmerkungen

Mit dieser Schnittstelle können TwinCAT Taskinformationen abgefragt und verwendet werden.

Siehe Beispiel30: Zeitmessung [▶ 256]

Sehen Sie dazu auch

- Methode ITcRTimeTask::GetCpuAccount() [▶ 150]

12.4.11.1 Methode ITcRTimeTask::GetCpuAccount()

Abfrage des CPU-Kontos eines TwinCAT Tasks.

Syntax

```
virtual HRESULT TCOMAPI GetCpuAccount(PULONG pAccount)=0;
```

Parameter

pAccount: (Typ: PULONG) [out] TwinCAT Task CPU-Konto ist in diesem Parameter gespeichert.

Rückgabewert

E_POINTER wenn Parameter pAccount gleich NULL, sonst S_OK

Beschreibung

Mit Hilfe der GetCpuAccount() Methode kann die aktuelle, für den Task aufgewendete Rechenzeit abgefragt werden.

Code-Ausschnitt, der die Verwendung von GetCpuAccount() zeigt, z.B. innerhalb einer ITcCyclic::CycleUpdate() Methode:

```
// CPU account in 100 ns interval
ITcRTimeTaskPtr spRTimeTask = ipTask;
ULONG nCpuAccountForComputeSomething = 0;
if (spRTimeTask != NULL)
{
ULONG nStart = 0;
hr = FAILED(hr) ? hr : spRTimeTask->GetCpuAccount(&nStart);

ComputeSomething();

ULONG nStop = 0;
hr = FAILED(hr) ? hr : spRTimeTask->GetCpuAccount(&nStop);

nCpuAccountForComputeSomething = nStop - nStart;
}
```

12.4.12 Schnittstelle ITcTask

Abfrage von Zeitstempel und taskspezifischen Informationen eines TwinCAT Tasks.

Syntax

```
TCOM_DECL_INTERFACE("02000002-0000-0000-e000-000000000064", ITcTask)
struct __declspec(novtable) ITcTask : public ITcUnknown
```

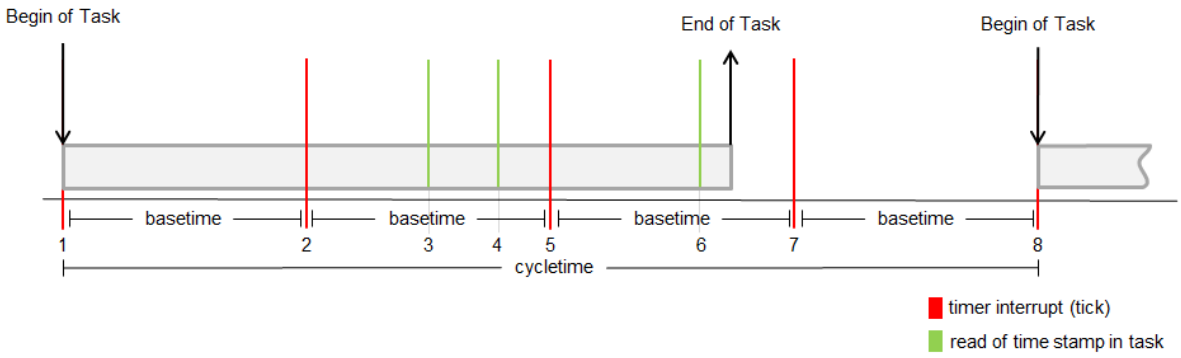
Benötigtes include: `TcRtInterfaces.h`

Methoden

Sym-bol	Name	Beschreibung
	GetCycleCounter [▶ 153]	Anzahl Taskzyklen seit Taskstart abfragen.
	GetCycleTime [▶ 153]	Abfrage der Taskzyklenzeit in Nanosekunden, d.h. Zeit zwischen „begin of task“ und nächstem „begin of task“
	GetPriority [▶ 151]	Abfrage der Taskpriorität
	GetCurrentSysTime [▶ 152]	Abfrage der Zeit bei Taskzyklusstart in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).
	GetCurrentDcTime [▶ 152]	Abfrage der Distributed-Clock-Zeit bei Taskzyklusbeginn in Nanosekunden seit dem 1. Januar 2000.
	GetCurPentiumTime [▶ 153]	Abfrage der Zeit bei Methodenaufruf in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Anmerkungen

Mit der ITcTask Schnittstelle kann die Zeit im Echtzeitkontext gemessen werden.



Reference	Function (ITcTask)	Unit	Zerotime	read time stamps at		
				3	4	6
Distributed Clock master (EtherCAT, Sercos,...)	GetCurrentSysTime	100ns	01.01.1601	1	1	1
	GetCurrentDcTime	1ns	01.01.2000	1	1	1
Processor Clock	GetCurPentiumTime	100ns	01.01.1601	3	4	6

Sehen Sie dazu auch

- Schnittstelle ITcRTimeTask [▶ 149]
- Beispiel30: Zeitmessung [▶ 256]

12.4.12.1 Methode ITcTask:GetPriority

Abfrage der Taskpriorität

Syntax

```
virtual HRESULT TCOMAPI GetPriority(PULONG
pPriority)=0;
```

Parameter

pPriority: (Typ: PULONG) [out] Prioritätswert des Tasks ist in diesem Parameter gespeichert.

Rückgabewert

E_POINTER wenn Parameter pPriority gleich NULL, sonst S_OK

Beschreibung

Beispiel30: Zeitmessung [▶ 256] veranschaulicht die Verwendung dieser Methode.

12.4.12.2 Methode ITcTask:GetCurrentSysTime

Abfrage der Zeit bei Taskzyklusstart in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Syntax

```
virtual HRESULT TCOMAPI GetCurrentSysTime(PLONGLONG
pSysTime)=0;
```

Parameter

pSysTime: (Typ: PLONGLONG) [out] in diesem Parameter ist die aktuelle Systemzeit zu Beginn des Taskzyklus gespeichert.

Rückgabewert

E_POINTER wenn Parameter pSysTime gleich NULL, sonst S_OK.

Beschreibung

Beispiel30: Zeitmessung [▶ 256] veranschaulicht die Verwendung dieser Methode.

12.4.12.3 Methode ITcTask:GetCurrentDcTime

Abfrage der Distributed-Clock-Zeit bei Taskzyklusbeginn in Nanosekunden seit dem 1. Januar 2000.

Syntax

```
virtual HRESULT TCOMAPI GetCurrentDcTime(PLONGLONG
pDcTime)=0;
```

Parameter

pDcTime: (Typ: PLONGLONG) [out] in diesem Parameter ist die Distributed-Clock-Zeit zu Beginn des Taskzyklus gespeichert.

Rückgabewert

E_POINTER wenn Parameter pDcTime gleich NULL, sonst S_OK.

Beschreibung

Beispiel30: Zeitmessung [▶ 256] veranschaulicht die Verwendung dieser Methode.

12.4.12.4 Methode ITcTask:GetCurPentiumTime

Abfrage der Zeit bei Methodenaufruf in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Syntax

```
virtual HRESULT TCOMAPI GetCurPentiumTime(PLONGLONG
pCurTime)=0;
```

Parameter

pCurTime: (Typ: PLONGLONG) [out] in diesem Parameter wird die aktuelle Zeit (UTC) in 100 Nanosekunden-Intervallen seit dem 1. Januar 1601 gespeichert

Rückgabewert

E_POINTER wenn Parameter pCurTime gleich NULL, sonst S_OK

Beschreibung

Beispiel30: Zeitmessung [▶ 256] veranschaulicht die Verwendung dieser Methode.

12.4.12.5 Methode ITcTask:GetCycleCounter

Anzahl Taskzyklen seit Taskstart abfragen.

Syntax

```
virtual HRESULT TCOMAPI GetCycleCounter(PULONGLONG
pCnt)=0;
```

Parameter

pCnt: (Typ: PULONGLONG) [out] die Anzahl von Taskzyklen seit Task gestartet wurde, wird in diesem Parameter gespeichert

Rückgabewert

E_POINTER wenn Parameter pCnt gleich NULL, ansonsten S_OK

Beschreibung

Beispiel30: Zeitmessung [▶ 256] veranschaulicht die Verwendung dieser Methode.

12.4.12.6 Method ITcTask:GetCycleTime

Abfrage der Taskzyklenzeit in Nanosekunden, d.h. Zeit zwischen „begin of task“ und nächstem „begin of task“

Syntax

```
virtual HRESULT TCOMAPI GetCycleTime(PULONG
pCycleTimeNS)=0;
```

Parameter

pCycleTimeNS: (Typ: PULONG) [out] in diesem Parameter wird die konfigurierte Taskzyklenzeit in Nanosekunden gespeichert.

Rückgabewert

E_POINTER wenn Parameter pCnt gleich NULL, sonst S_OK

Beschreibung

Beispiel30: Zeitmessung [▶ 256] veranschaulicht die Verwendung dieser Methode.

12.4.13 Schnittstelle ITcTaskNotification

Führt einen Callback aus, wenn die Zykluszeit beim vorherigem Zyklus überschritten wurde.
Diese Schnittstelle stellt vergleichbare Funktionalitäten wie SPS PlcTaskSystemInfo->CycleTimeExceeded zur Verfügung.

Syntax

```
TCOM_DECL_INTERFACE("9CDE7C78-32A0-4375-827E-924B31021FCD", ITcTaskNotification) struct __decl-
spec(novtable) ITcTaskNotification: public ITcUnknown
```

Benötigtes include: `TcRtInterfaces.h`

Methoden

Symbol	Name	Beschreibung
	NotifyCycleTimeExceeded	Wird aufgerufen, wenn die Zykluszeit überschritten wurde.

Anmerkungen

Beachten Sie, dass der Callback nicht während den Berechnungen, sondern am Ende des Zyklus stattfindet. Also bietet diese Methode keinen Mechanismus, um die Berechnungen sofort zu stoppen.

Sehen Sie dazu auch

- Methode ITcTaskNotification::NotifyCycleTimeExceeded() [▶ 154]
- Schnittstelle ITcFileAccess [▶ 133]
- Beispiel20a: FileIO-Cyclic Read / Write [▶ 250]

12.4.13.1 Methode ITcTaskNotification::NotifyCycleTimeExceeded()

Wird aufgerufen, wenn die Zykluszeit zuvor abgelaufen ist

Syntax

```
virtual HRESULT TCOMAPI NotifyCycleTimeExceeded ();
```

Parameter

ipTask: (Typ: ITcTask) verweist auf den aktuellen Task-Kontext.

context: (Typ: ULONG_PTR) Kontext

Rückgabewert

Typ: HRESULT

Gibt S_OK zurück, wenn Dateioperation abgeschlossen ist

Beschreibung

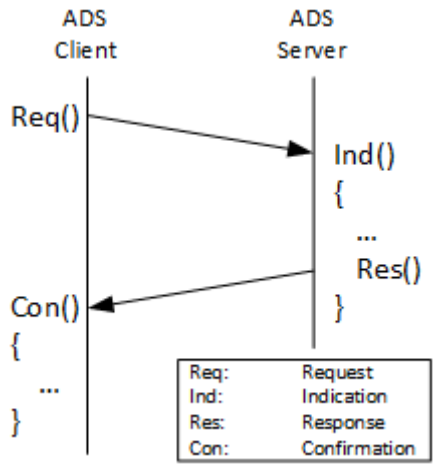
Wird aufgerufen, wenn die Zykluszeit vorher überschritten war. Also nicht sofort bei Zeitüberschreitung, sondern danach.

Sehen Sie dazu auch

- ADS Return Codes [▶ 260]

12.5 ADS-Kommunikation

Auf Client-Server-Prinzip basierende ADS (Abbildung 1). Eine ADS-Abfrage ruft die entsprechenden Indikationsmethoden auf der Serverseite auf. Die ADS-Antwort ruft die entsprechende Bestätigungsmethode auf der Client-Seite auf.



ADS Client-Server-Sequenz

In diesem Abschnitt werden sowohl die ausgehende, als auch die eingehende ADS-Kommunikation für TwinCAT 3 C++ Module beschrieben.

ADS-Befehlssatz	Beschreibung
AdsReadDeviceInfo [▶ 156]	Mit diesem Befehl können die allgemeinen Geräteinformationen gelesen werden.
AdsRead [▶ 158]	ADS-Lesebefehl, um Daten von einem ADS-Gerät abzufragen.
AdsWrite [▶ 160]	ADS-Schreibbefehl, um Daten an ein ADS-Gerät zu übergeben.
AdsReadState [▶ 164]	ADS-Befehl, um den Zustand von einem ADS-Gerät abzufragen.
AdsWriteControl [▶ 166]	ADS-Steuerungsbefehl, um den Zustand von einem ADS-Gerät zu ändern.
AdsAddDeviceNotification [▶ 168]	Variable beobachten. Der Client wird bei einem Ereignis informiert.
AdsDelDeviceNotification [▶ 170]	Entfernt die Variable, die zuvor verbunden war.
AdsDeviceNotification [▶ 172]	Wird für die Übermittlung des Geräte-Notification-Ereignisses verwendet.
AdsReadWrite [▶ 162]	ADS-Schreib-/Lesebefehl. Mit einem Aufruf werden Daten zu einem ADS-Gerät übermittelt (Write) und dessen Antwortdaten gelesen (Read).

Die ADS Return Codes [▶ 260] gelten für die gesamte ADS-Kommunikation.

Schauen Sie sich als Einstieg das Beispiel07: Empfang von ADS Notifications [▶ 215]

12.5.1 AdsReadDeviceInfo

12.5.1.1 AdsReadDeviceInfoReq

Die Methode AdsDeviceInfoReq ermöglicht das Übermitteln eines ADS DeviceInfo Befehls zum Auslesen von Identifizierung und Versionsnummer eines ADS Servers.
AdsReadDeviceInfoCon wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsReadDeviceInfoReq( AmsAddr& rAddr, ULONG invokeId );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.1.2 AdsReadDeviceInfoInd

Die Methode AdsDeviceInfoInd verweist auf einen ADS DeviceInfo Befehl zum Lesen der Identifizierung und Versionsnummer eines ADS Servers. Anschließend muss AdsReadDeviceInfoRes [▶ 156] aufgerufen werden.

Syntax

```
void    AdsReadDeviceInfoInd( AmsAddr& rAddr, ULONG invokeId );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

Rückgabewert

void

12.5.1.3 AdsReadDeviceInfoRes

Die Methode AdsReadDeviceInfoRes sendet eine ADS Read Device Info. AdsReadDeviceInfoCon bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int    AdsReadDeviceInfoRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, CHAR name[ADS_FIXEDNAME-
SIZE], AdsVersion version );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

name: (Typ: char[ADS_FIXEDNAMESIZE]) [in] enthält den Namen des Geräts.

version: (Typ: AdsVersion) [in] struct von Build (int), Revision (Byte) und Version (Byte) des Geräts

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes

12.5.1.4 AdsReadDeviceInfoCon

Die Methode AdsReadDeviceInfoCon ermöglicht den Empfang von einer ADS-Lesebestätigung einer Geräteinformation. Das empfangende Modul muss diese Methode bereitstellen. Vorher muss das Pendant AdsReadDeviceInfoReq [▶ 156]aufgerufen werden.

Syntax

```
void AdsReadDeviceInfoCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult,
CHAR name[ADS_FIXEDNAMESIZE], AdsVersion version );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

name: (Typ: char[ADS_FIXEDNAMESIZE]) [in] enthält den Namen des Geräts.

version: (Typ: AdsVersion) [in] struct von Build (int), Revision (Byte) und Version (Byte) des Geräts

Rückgabewert

void

12.5.2 AdsRead

12.5.2.1 AdsReadReq

Die Methode AdsReadReq ermöglicht das Senden eines ADS-Lesebefehls für die Datenübertragung von einem ADS-Gerät.
Beim Eingang der Antwort wird AdsReadCon [▶ 159]aufgerufen.

Syntax

```
int AdsReadReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData).

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.2.2 AdsReadInd

Die Methode AdsReadInd ermöglicht den Empfang von einer ADS-Leseanforderung. Die AdsReadRes [▶ 159] muss aufgerufen werden, um das Ergebnis zu senden.

Syntax

```
void    AdsReadInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData).

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes

12.5.2.3 AdsReadRes

Die Methode AdsReadRes ermöglicht das Senden von einer ADS-Leseantwort. AdsReadCon [▶ 159] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int    AdsReadRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Lesebefehls. Siehe AdsStatuscodes [▶ 260]

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die Daten sich befinden.

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes

12.5.2.4 AdsReadCon

Die Methode AdsReadCon ermöglicht den Empfang von einer ADS-Lesebestätigung. Das empfangende Modul muss diese Methode bereitstellen.
Das Pendant AdsReadReq [▶ 158] bildet muss vorher aufgerufen worden sein.

Syntax

```
void AdsReadCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Lesebefehls. Siehe AdsStatuscodes [▶ 260]

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

Rückgabewert

void

12.5.3 AdsWrite

12.5.3.1 AdsWriteReq

Die Methode AdsWriteReq ermöglicht das Senden eines ADS-Schreibbefehls, für Übertragung von Daten an ein ADS-Gerät.
Beim Eingang der Antwort wird AdsWriteCon [▶ 161] aufgerufen.

Syntax

```
int AdsWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.3.2 AdsWriteInd

Die Methode AdsWriteInd gibt einen ADS-Schreibbefehl an, um Daten zu einem ADS-Gerät zu übermitteln.
Die AdsWriteRes [▶ 161] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void    AdsWriteInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

Rückgabewert

void

Sehen Sie dazu auch

ADS Return Codes [▶ 260]

12.5.3.3 AdsWriteRes

Die Methode AdsWriteRes sendet eine ADS-Schreibantwort. AdsWriteCon [▶ 161] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 260]

12.5.3.4 AdsWriteCon

Die Methode AdsWriteCon ermöglicht den Empfang einer ADS-Schreibbestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsWriteReq [▶ 160] bildet das Gegenstück und muss zuvor aufgerufen worden sein.

Syntax

```
void AdsWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

Rückgabewert

void

12.5.4 AdsReadWrite

12.5.4.1 AdsReadWriteReq

Die Methode AdsReadWriteReq ermöglicht das Senden eines ADS-Lese-/Schreibbefehls für die Datenübergabe an ein und von einem ADS-Gerät. Die AdsReadWriteCon [▶ 164] wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsReadWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbReadLength, ULONG cbWriteLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbReadLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData)

cbWriteLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.4.2 AdsReadWriteInd

Die Methode AdsReadWriteInd gibt einen ADS-Lese-/Schreibbefehl an, um Daten zu einem ADS-Gerät zu übermitteln. Die AdsReadWriteRes [▶ 165] muss aufgerufen werden, um das Ergebnis zu senden.

Syntax

```
void    AdsReadWriteInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup,
ULONG indexOffset, ULONG cbReadLength, ULONG cbWriteLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbReadLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu lesenden Daten (pData)

cbWriteLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der zu schreibenden Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

Rückgabewert

void

12.5.4.3 AdsReadWriteRes

Die Methode AdsReadWriteRes ermöglicht den Empfang von einer ADS-Lese-/Schreibbestätigung. AdsReadWriteCon [▶ 164] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int    AdsReadWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pDa-
ta );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes

12.5.4.4 AdsReadWriteCon

Die Methode AdsReadWriteCon ermöglicht den Empfang von einer ADS-Lese-/Schreibbestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsReadWriteReq [▶ 162] bildet das Gegenstück und muss zuvor aufgerufen.

Syntax

```
void AdsReadWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der gelesenen Daten (pData).

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem sich die Daten befinden.

Rückgabewert

void

12.5.5 AdsReadState

12.5.5.1 AdsReadStateReq

Die Methode AdsReadStateReq ermöglicht die Übermittlung eines ADS-Statuslesebefehls für das Lesen des ADS- und des Gerätezustands von einem ADS Server. Beim Eingang der Antwort wird AdsReadStateCon [▶ 165] aufgerufen.

Syntax

```
int AdsReadStateReq(AmsAddr& rAddr, ULONG invokeId);
```

Parameter

rAddr: (Typ: AmsAddr) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.5.2 AdsReadStateInd

Die Methode AdsReadStateInd zeigt einen ADS-Statuslesebefehl an für das Lesen des ADS-Zustands und des Gerätezustands von einem ADS-Gerät. Die AdsReadStateRes [▶ 165] muss aufgerufen werden, um das Ergebnis zu senden.

Syntax

```
void    AdsReadStateInd( AmsAddr& rAddr, ULONG invokeId );
```

Parameter

rAddr: (Typ: AmsAddr) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

Rückgabewert

void

12.5.5.3 AdsReadStateRes

Die Methode AdsWriteRes ermöglicht das Senden von einer ADS-Statusleseantwort. AdsReadStateCon [▶ 165] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int    AdsReadStateRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT devi-
ceState );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

adsState: (Typ: USHORT) [in] enthält den ADS-Zustand des Geräts

deviceState: (Typ: USHORT) [in] enthält den Gerätestatus des Geräts

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes

12.5.5.4 AdsReadStateCon

Die Methode AdsWriteCon ermöglicht den Empfang von einer ADS-Statuslesebestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsReadStateReq [▶ 164] bildet das Gegenstück und muss zuvor aufgerufen.

Syntax

```
void AdsReadStateCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT device-
State );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

adsState: (Typ: USHORT) [in] enthält den ADS-Zustand des Geräts

deviceState: (Typ: USHORT) [in] enthält den Gerätestatus des Geräts

Rückgabewert

void

12.5.6 AdsWriteControl

12.5.6.1 AdsWriteControlReq

Mit der Methode AdsWriteControlReq kann ein ADS-Schreibsteuerungsbefehl zur Änderung des ADS- und Gerätestatus eines ADS Servers gesendet werden. Beim Eingang der Antwort wird AdsWriteControlCon [▶ 168] aufgerufen.

Syntax

```
int AdsWriteControlReq( AmsAddr& rAddr, ULONG invokeId, USHORT adsState,
    USHORT deviceState, ULONG cbLength, PVOID pData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

adsState: (Typ: USHORT) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

deviceState: (Typ: USHORT) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der Daten (pData)

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.6.2 AdsWriteControlInd

Mit der Methode AdsWriteControlInd kann ein ADS-Schreibsteuerungsbefehl zur Änderung des ADS- und Gerätestatus eines ADS-Geräts gesendet werden. Die AdsWriteControlRes [▶ 167] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void    AdsWriteControlInd( AmsAddr& rAddr, ULONG invokeId, USHORT adsState, USHORT deviceState,
ULONG cbLength, PVOID pDeviceData );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

adsState: (Typ: USHORT) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

deviceState: (Typ: USHORT) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

cbLength: (Typ: ULONG) [in] enthält die Länge, in Bytes, der Daten (pData)

pData: (Typ: PVOID) [in] Zeiger auf den Datenpuffer, in dem die geschriebenen Daten sich befinden.

Rückgabewert

void

12.5.6.3 AdsWriteControlRes

Die Methode AdsWriteControlRes ermöglicht das Senden von einer ADS-Schreibsteuerungsantwort. AdsWriteControlCon [▶ 168] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsWriteControlRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 260]

12.5.6.4 AdsWriteControlCon

Die Methode AdsWriteCon ermöglicht den Empfang von einer ADS-Schreibsteuerungsbestätigung. Das empfangende Modul muss diese Methode bereitstellen.
AdsWriteControlReq [▶ 166] bildet das Gegenstück und muss zuvor aufgerufen

Syntax

```
void AdsWriteControlCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

Rückgabewert

void

12.5.7 AdsAddDeviceNotification

12.5.7.1 AdsAddDeviceNotificationReq

Die Methode AdsAddDeviceNotificationReq ermöglicht das Senden eines ADS-Hinzufügen-Geräte-Notification-Befehls, um eine Geräte-Notification zu einem ADS-Gerät hinzuzufügen.
AdsAddDeviceNotificationCon [▶ 170] wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsAddDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOff-
set,
    AdsNotificationAttrib noteAttrib);
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

noteAttrib: (Typ: AdsNotificationAttrib) [in] enthält die Spezifikation der Notification-Parameter (cbLength, TransMode, MaxDelay).

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.7.2 AdsAddDeviceNotificationInd

Die Methode AdsAddDeviceNotificationInd sollte das Senden von AdsDeviceNotification [▶ 172] ermöglichen. Die AdsAddDeviceNotificationRes [▶ 169] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void    AdsAddDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG inde-
xOffset, AdsNotificationAttrib noteAttrib );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

indexGroup: (Typ: ULONG) [in] enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

indexOffset: (Typ: ULONG) [in] enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

noteAttrib: (Typ: AdsNotificationAttrib) [in] enthält die Spezifikation der Notification-Parameter (cbLength, TransMode, MaxDelay).

Rückgabewert

void

12.5.7.3 AdsAddDeviceNotificationRes

Die Methode AdsAddDeviceNotificationRes ermöglicht das Senden von einer ADS-Gerät-Hinzufügen-Notification-Antwort. AdsAddDeviceNotificationCon [▶ 170] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes **Handle**: (Typ: ULONG) [in] Handle auf generierte Geräte-Notification

Rückgabewert

void

12.5.7.4 AdsAddDeviceNotificationCon

Die Methode AdsAddDeviceNotificationCon bestätigt eine ADS-Gerät-Hinzufügen-Notification-Anforderung. AdsAddDeviceNotificationReq [▶ 168] bildet das Gegenstück und muss zuvor aufgerufen.

Syntax

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes Handle

handle: (Typ: ULONG) [in] Handle auf generierte Geräte-Notification

Rückgabewert

void

12.5.8 AdsDelDeviceNotification

12.5.8.1 AdsDelDeviceNotificationReq

Die Methode AdsDelDeviceNotificationReq ermöglicht das Senden eines ADS-Gerät-Löschen-Notification-Befehls, um eine Geräte-Notification von einem ADS-Gerät zu entfernen. Die AdsDelDeviceNotificationCon [▶ 171] wird beim Eingang der Antwort aufgerufen.

Syntax

```
int AdsDelDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG hNotification );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen

hNotification: (Typ: ULONG) [in] enthält das Handle der zu entfernenden Notification

Rückgabewert

Typ: int

Fehlercode - Siehe AdsStatuscodes [▶ 260]

12.5.8.2 AdsDelDeviceNotificationInd

Die Methode AdsAddDeviceNotificationCon ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification-Bestätigung. Das empfangende Modul muss diese Methode bereitstellen. Die AdsDelDeviceNotificationRes [▶ 171] muss aufgerufen werden, um den Vorgang zu bestätigen.

Syntax

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls. Siehe AdsStatuscodes [▶ 260]

Rückgabewert

void

12.5.8.3 AdsDelDeviceNotificationRes

Die Methode AdsAddDeviceNotificationRes ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification. AdsDelDeviceNotificationCon [▶ 171] bildet das Gegenstück und wird anschließend aufgerufen.

Syntax

```
int AdsDelDeviceNotificationRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Befehls. Siehe AdsStatuscodes [▶ 260]

Rückgabewert

Int

Gibt das Ergebnis des ADS-Befehls zurück. Siehe AdsStatuscodes [▶ 260]

12.5.8.4 AdsDelDeviceNotificationCon

Die Methode AdsAddDeviceNotificationCon ermöglicht den Empfang von einer ADS-Gerät-Löschen-Notification-Bestätigung. Das empfangende Modul muss diese Methode bereitstellen. AdsDelDeviceNotificationReq [▶ 170] bildet das Gegenstück und muss zuvor aufgerufen.

Syntax

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des gesendeten Befehls, die InvokeId wird vom Quellgerät spezifiziert und dient der Identifizierung der Befehle.

nResult: (Typ: ULONG) [in] enthält das Ergebnis des ADS-Schreibbefehls, siehe AdsStatuscodes [▶ 260].

Rückgabewert

void

12.5.9 AdsDeviceNotification

12.5.9.1 AdsDeviceNotificationReq

Die Methode AdsAddDeviceNotificationReq ermöglicht das Senden einer ADS-Geräte-Notification, um ein ADS-Gerät zu informieren. Die AdsDeviceNotificationInd [▶ 172] wird auf dem Gegenstück aufgerufen.

Syntax

```
int    AdsDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength, AdsNotificationS-
tream notifications[] );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis der Geräte-Notification-Anzeige.

notifications[]: (Typ: AdsNotificationStream) [in] enthält Informationen der Geräte-Notification(s).

Rückgabewert

Typ: int

ADS Return Code - siehe AdsStatuscodes [▶ 260]

12.5.9.2 AdsDeviceNotificationInd

Die Methode AdsDeviceNotificationInd ermöglicht den Empfang von einer ADS-Geräte-Notification-Anzeige. Das empfangende Modul muss diese Methode bereitstellen. Die AdsDeviceNotificationCon [▶ 173] muss zwecks Bestätigung dieser Anzeige aufgerufen werden.

Syntax

```
void AdsDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength,
AdsNotificationStream* pNotifications );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom antwortenden ADS Server

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

cbLength: (Typ: ULONG) [in] enthält die Länge von pNotifications

pNotifications: (Typ: AdsNotificationStream*) [in] Zeiger auf die Notifications. Dieses Array besteht aus AdsStampHeader mit Notification Handle und Daten via AdsNotificationSample.

Rückgabewert

void

12.5.9.3 AdsDeviceNotificationCon

Die Methode AdsAddDeviceNotificationCon bestätigt den Empfang einer ADS-Geräte-Notification. AdsDeviceNotificationReq [▶ 172] bildet das Gegenstück und muss zuvor aufgerufen.

Syntax

```
void    AdsDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

Parameter

rAddr: (Typ: AmsAddr&) [in] Struktur mit NetId und Portnummer vom ADS Server.

invokeId: (Typ: ULONG) [in] Handle des Befehls, der gesendet wird. Die InvokeId wird vom Quellgerät spezifiziert und wird für die Identifizierung der Befehle herangezogen.

nResult: (Typ: ULONG) [in] enthält das Ergebnis der Geräte-Notification-Anzeige.

Rückgabewert

void

12.6 Mathematische Funktionen

TwinCAT hat eigene mathematische Funktionen implementiert, weil die math.h Implementierung von Microsoft nicht echtzeitfähig ist.
Diese Funktionen sind in fpu87.h deklariert, einem Teil von TwinCAT SDK.

Bereitgestellte Methoden

Mathematische Funktionen

	Name	Beschreibung
	sqr_	Quadriert.
	sqrt_	Berechnet die Quadratwurzel.
	sin_	Berechnet den Sinus.
	cos_	Berechnet den Kosinus.
	tan_	Berechnet den Tangens.
	atan_	Berechnet den Winkel, dessen Tangens der angegebene Wert ist.
	atan2_	Berechnet den Winkel, dessen Tangens der Quotient zweier angegebener Werte ist.
	asin_	Berechnet den Winkel, dessen Sinus der angegebene Wert ist.
	acos_	Berechnet den Winkel, dessen Kosinus der angegebene Wert ist.
	exp_	Berechnet e hoch angegebene Potenz.
	log_	Berechnet den Logarithmus eines angegebenen Werts.
	log10_	Berechnet den Logarithmus zur Basis 10 eines angegebenen Werts.
	fabs_	Berechnet den Absolutwert.
	fmod_	Berechnet den Restbetrag.
	ceil_	Berechnet die kleinste Integerzahl die größer oder gleich der angegebenen Zahl ist.
	floor_	Berechnet die größte Integerzahl die kleiner oder gleich der angegebenen Zahl ist.
	pow_	Berechnet eine angegebene Zahl hoch angegebener Potenz.
	sincos_	Berechnet den Sinus und den Kosinus von x.
	fmodabs_	Berechnet den Absolutbetrag, der die euklidische Definition der mod-Operation erfüllt.
	round_	Berechnet einen Wert und rundet ihn auf den nächsten Integer.
	round_digits_	Berechnet einen gerundeten Wert mit einer festgelegten Anzahl Dezimalstellen.
	coubic_	Berechnet die Kubikzahl.
	ldexp_	Berechnet eine reelle Zahl (double) aus Mantisse und Exponent.
	ldexpf_	Berechnet eine reelle Zahl (float) aus Mantisse und Exponent.
	sinh_	Berechnet den Hyperbelsinus des angegebenen Winkels.
	cosh_	Berechnet den Hyperbelkosinus des angegebenen Winkels.
	tanh_	Berechnet den Hyperbeltangens des angegebenen Winkels.
	Hilfsfunktionen	
	finite_	Ermittelt, ob der angegebene Wert endlich ist.
	isnan_	Ermittelt, ob der gegebene Wert keine Zahl ist (NAN, „not a number“).

	Name	Beschreibung
	rands_	Berechnet eine Pseudo-Zufallszahl zwischen 0 und 32767. Der Parameter holdrand wird zufällig gesetzt und bei jedem Aufruf geändert.

Anmerkungen

Die Funktionen haben die Endung „_“ (Unterstrich), um sie als TwinCAT Implementierung zu kennzeichnen. Die meisten sind analog math.h von Microsoft konzipiert, nur für den Datentyp Double.

Siehe auch

MSDN Dokumentation von Funktionen analog math.h.

12.7 Zeitfunktionen

TwinCAT bietet Funktionen für Zeitumwandlung.

Diese Funktionen werden in TcTimeConversion.h, die Teil von TwinCAT SDK ist, deklariert.

	Name	Beschreibung
	TcDayOfWeek	Ermittelt den Wochentag (Sonntag ist 0)
	TcIsLeapYear	Ermittelt, ob das gegebene Jahr ein Schaltjahr ist.
	TcDaysInYear	Ermittelt die Anzahl Tage im gegebenen Jahr
	TcDaysInMonth	Ermittelt die Anzahl Tage im gegebenen Monat
	TcSystemTimeToFileTime(const SYSTEMTIME* lpSystemTime, FILETIME *lpFileTime);	Wandelt die gegebene Systemzeit in eine Dateizeit um.
	TcFileTimeToSystemTime(const FILETIME *lpFileTime, SYSTEMTIME* lpSystemTime);	Wandelt die gegebene Dateizeit in eine Systemzeit um.
	TcSystemTimeToFileTime(const SYSTEMTIME* lpSystemTime, ULONGLONG& ul64FileTime);	Wandelt die gegebene Systemzeit in eine Dateizeit um (ULONGLONG Format)
	TcFileTimeToSystemTime(const ULONGLONG& ul64FileTime, SYSTEMTIME* lpSystemTime);	Wandelt die gegebene Dateizeit (ULONGLONG Format) in eine Systemzeit um.

12.8 STL / Container

Hinweis	**Nur x86 (32Bit)** Beachten Sie, dass die STL Funktion derzeit nur unter x86 / 32Bit verfügbar ist.

TwinCAT 3 C++ unterstützt STL bezüglich

- Vector
- Map
- List
- String

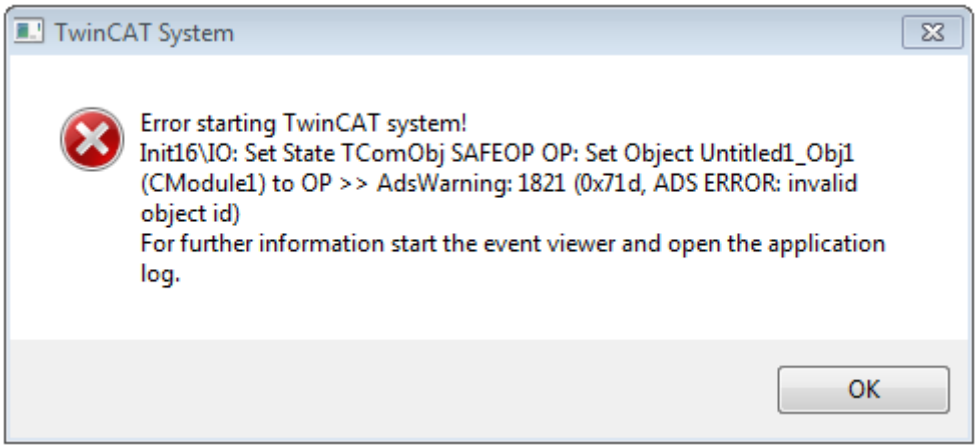
Speicherplatz wird in Abhängigkeit vom Zustand [▶ 27] alloziert:

- Beim Übergang von PreOp zu SafeOp (PS): vom non-paged Pool des Betriebssystems

- In Echtzeitkontext: von TwinCAT Routerspeicher

12.9 Fehlermeldungen - Verständnis

Es mag den Anschein haben, dass die Fehlermeldungen von TwinCAT überladen sind.


TwinCAT System
Error starting TwinCAT system!
Init16\IO: Set State TComObj SAFEOP OP: Set Object Untitled1_Obj1 (CModule1) to OP >> AdsWarning: 1821 (0x71d, ADS ERROR: invalid object id)
For further information start the event viewer and open the application log.
OK

Sie enthalten sehr ausführliche Informationen über den aufgetretenen Fehler.
So beschreibt z.B. der Screenshot oben Folgendes:

- Der Fehler trat beim Übergang von SAFE OP zu OP auf
- Das betroffene Objekt ist „Untitled1_Obj1“ (CModule1)
- Der Fehlercode 1821 / 0x71d zeigt an, dass die Objekt-Id ungültig ist

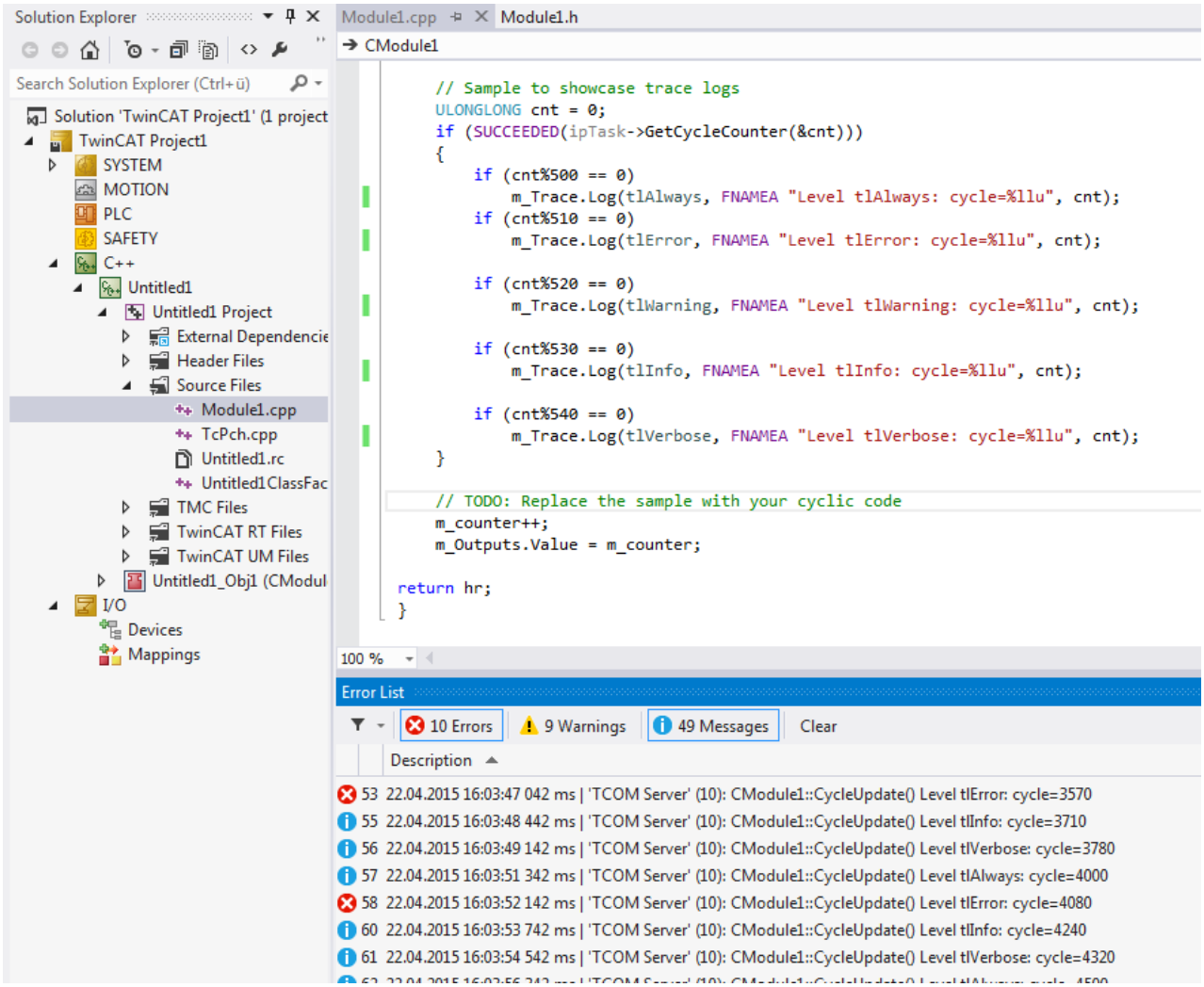
Die Folge ist, dass man die Methode „SetObjStateSP()“, die für diesen Übergang verantwortlich ist, genauer untersuchen muss. Im Falle des generierten Standardcodes sieht man, dass das Hinzufügen des Moduls dort stattfindet.

Der Grund für das Auftreten dieses Fehlers besteht darin, dass diesem Modul kein Task zugewiesen wurde, also kann das Modul keinen Task haben, in dem es ausgeführt wird.

12.10 Meldungen aufzeichnen

Übersicht

TwinCAT 3 C++ bietet Verfolgungsfähigkeiten um Rückverfolgung aus C++ Code in die TwinCAT 3 Visual Studio Workbench einfach zu ermöglichen.



Syntax

Die Syntax zur Aufzeichnung von Meldungen ist folgende:

```
m_Trace.Log(TLEVEL, FNMACRO"A message", …);
```

Mit diesen Eigenschaften:

- `TLEVEL` kategorisiert eine Meldung in eine von fünf Ebenen.
  Das Aufzeichnen der höheren Ebene beinhaltet immer auch das Aufzeichnen der unteren Ebenen:
  D.h. eine auf Ebene „tlWarning“ klassifizierte Meldung wird auftreten mit Ebene „tlAlways“, „tlError“ und „tlWarning“ - sie wird die „tlInfo“ und „tlVerbose“ Meldungen NICHT aufzeichnen.

Level 0	tlAlways
Level 1	tlError
Level 2	tlWarning
Level 3	tlInfo
Level 4	tlVerbose

- `FNMACRO` kann verwendet werden, um den Funktionsnamen vor die zu druckende Meldung zu setzen
  - FENTERA: Wird beim Eintritt in eine Funktion verwendet; druckt den Funktionsnamen gefolgt von „>>>“
  - FNAMEA: Wird innerhalb einer Funktion verwendet; druckt den Funktionsnamen
  - FLEAVEA: Wird beim Verlassen einer Funktion verwendet; druckt den Funktionsnamen gefolgt von „<<<“
- Formatspezifizierer

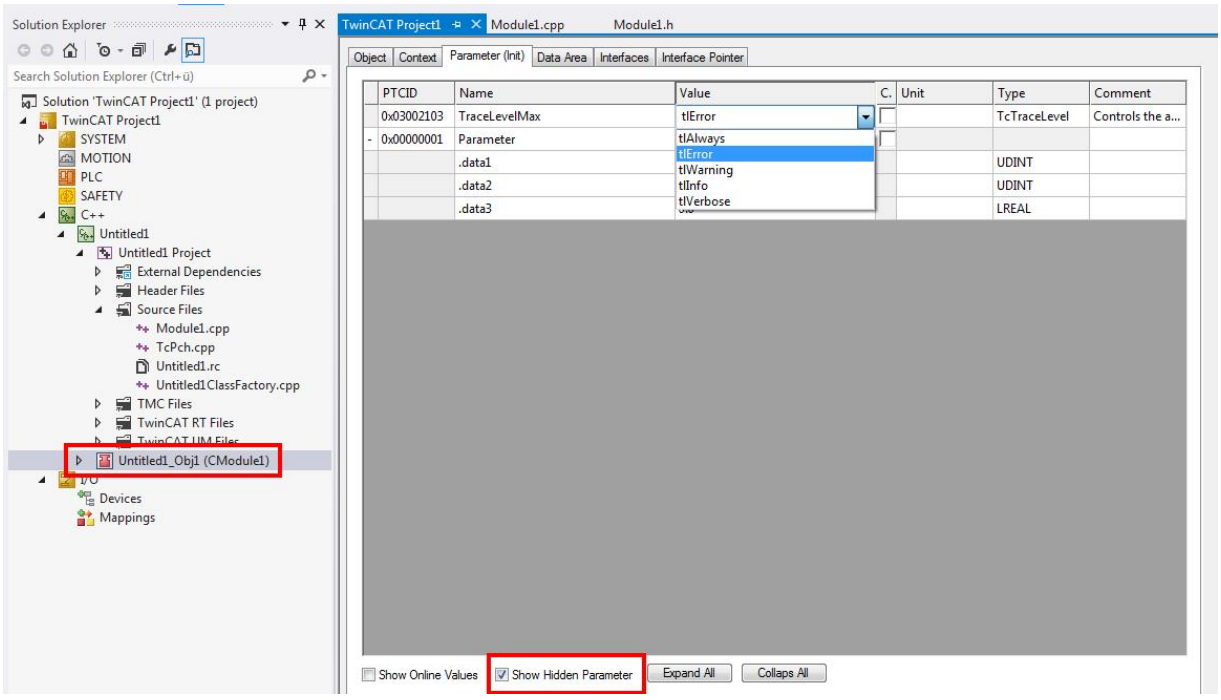
Beispiel

```
HRESULT CModule1::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    // Sample to showcase trace logs
    ULONGLONG cnt = 0;
    if (SUCCEEDED(ipTask->GetCycleCounter(&cnt)))
    {
        if (cnt%500 == 0)
            m_Trace.Log(tlAlways, FENTERA "Level tlAlways: cycle= %llu", cnt);

        if (cnt%510 == 0)
            m_Trace.Log(tlError, FENTERA "Level tlError: cycle=%llu", cnt);

        if (cnt%520 == 0)
            m_Trace.Log(tlWarning, FENTERA "Level tlWarning: cycle=%lld", cnt);

        if (cnt%530 == 0)
            m_Trace.Log(tlInfo, FENTERA "Level tlInfo: cycle=%llu", cnt);

        if (cnt%540 == 0)
            m_Trace.Log(tlVerbose, FENTERA "Level tlVerbose: cycle=%llu", cnt);
    }

    // TODO: Replace the sample with your cyclic code
    m_counter++;
    m_Outputs.Value = m_counter;

    return hr;
}
```

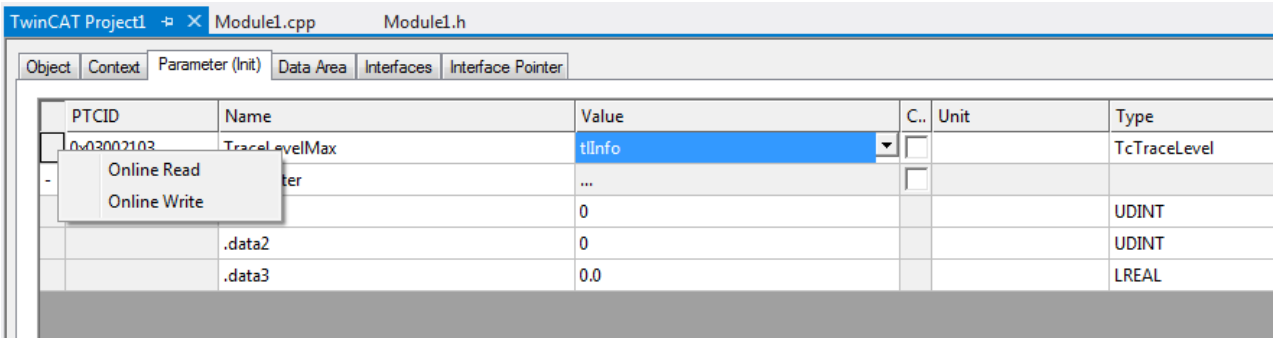
Verfolgungsebene verwenden

Auf Höhe der Modulinstanz besteht die Möglichkeit, die Verfolgungsebene vorzukonfigurieren.

1. Navigieren Sie zur Instanz des Moduls im Solution-Baum
2. Wählen Sie den Karteireiter „Parameter (Init)“ auf der rechten Seite.
3. Achten Sie darauf, dass sie „Show Hidden Parameters“ aktivieren
4. Wählen Sie die Verfolgungsebene
5. Um alles zu testen, wählen Sie die höchste Ebene „tlVerbose“.



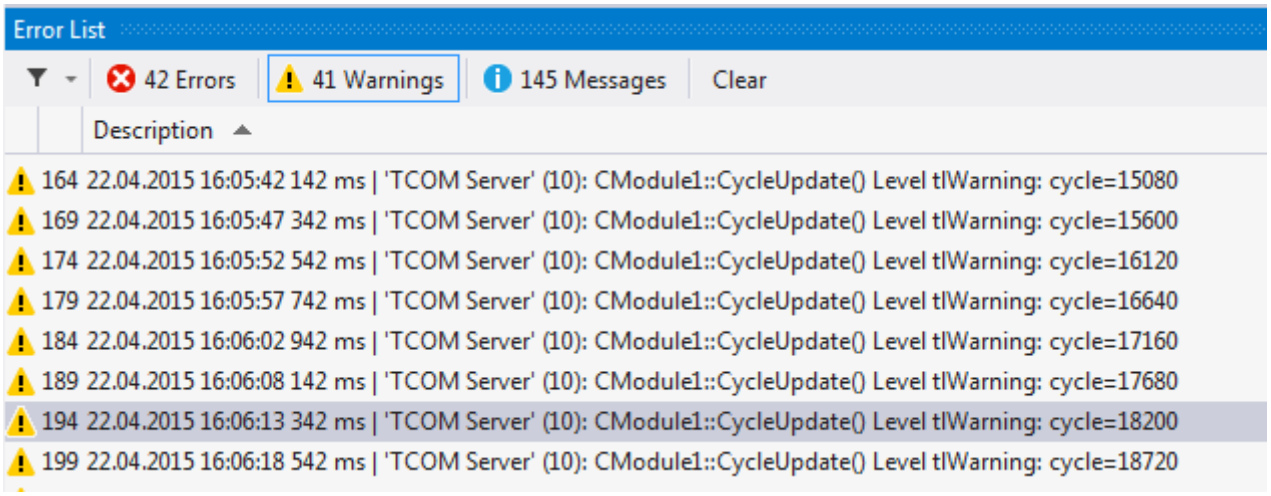
Alternativ dazu kann die Verfolgungsebene auch während der Laufzeit geändert werden, indem man zur Instanz geht, eine Ebene bei „Value“ für TraceLevelMax-Parameter auswählt, dann Rechtsklick vor der ersten Spalte und „Online Write“ auswählen.



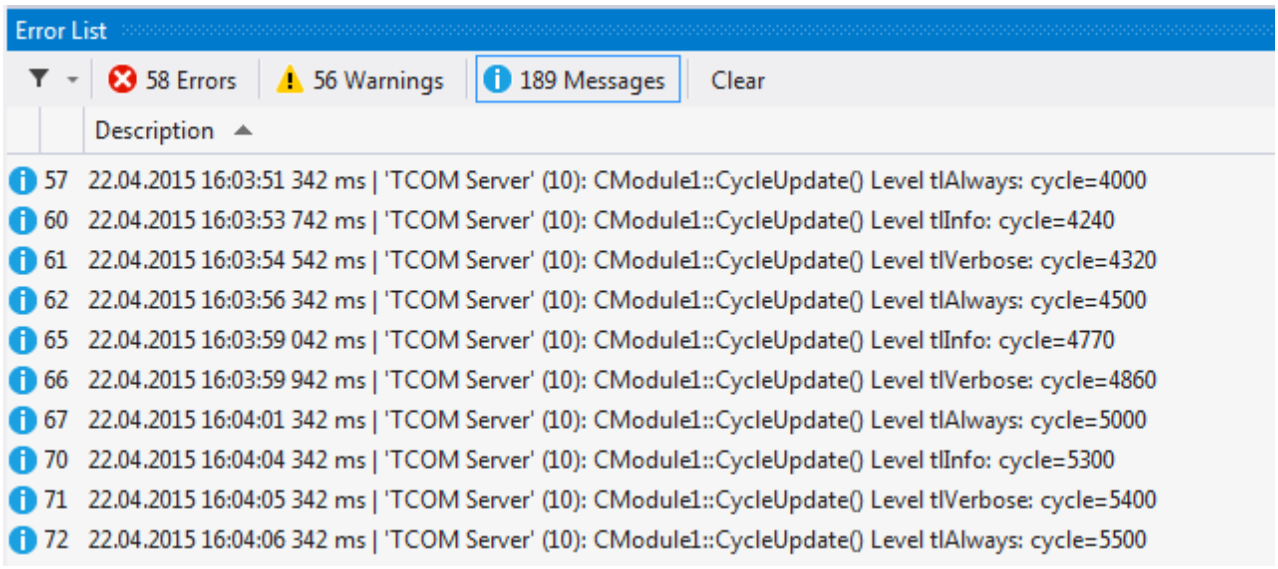
Meldungskategorien filtern

Visual Studio Error List ermöglicht das Filtern der Einträge nach deren Kategorie. Die drei Kategorien „Errors“, „Warnings“ und „Messages“ können unabhängig voneinander durch einfaches Umschalten der Tasten aktiviert oder deaktiviert werden.

In diesem Screenshot ist nur „Warnings“ aktiviert - „Errors“ und „Messages“ dagegen sind deaktiviert:


Error List
42 Errors
41 Warnings
145 Messages
Clear
Description
164 22.04.2015 16:05:42 142 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=15080
169 22.04.2015 16:05:47 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=15600
174 22.04.2015 16:05:52 542 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=16120
179 22.04.2015 16:05:57 742 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=16640
184 22.04.2015 16:06:02 942 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=17160
189 22.04.2015 16:06:08 142 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=17680
194 22.04.2015 16:06:13 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=18200
199 22.04.2015 16:06:18 542 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlWarning: cycle=18720

In diesem Screenshot ist nur „Messages“ aktiviert - „Errors“ und „Warnings“ dagegen sind für die Anzeige deaktiviert:


Error List
58 Errors
56 Warnings
189 Messages
Clear
Description
57 22.04.2015 16:03:51 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlAlways: cycle=4000
60 22.04.2015 16:03:53 742 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlInfo: cycle=4240
61 22.04.2015 16:03:54 542 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlVerbose: cycle=4320
62 22.04.2015 16:03:56 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlAlways: cycle=4500
65 22.04.2015 16:03:59 042 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlInfo: cycle=4770
66 22.04.2015 16:03:59 942 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlVerbose: cycle=4860
67 22.04.2015 16:04:01 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlAlways: cycle=5000
70 22.04.2015 16:04:04 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlInfo: cycle=5300
71 22.04.2015 16:04:05 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlVerbose: cycle=5400
72 22.04.2015 16:04:06 342 ms | 'TCOM Server' (10): CModule1::CycleUpdate() Level tlAlways: cycle=5500

13 How to...?

Eine Sammlung häufig gestellter Fragen zu allgemeinen Programmierbeispielen und dem Handling von TwinCAT C++ Modulen.

13.1 Ziellaufzeit auswählen

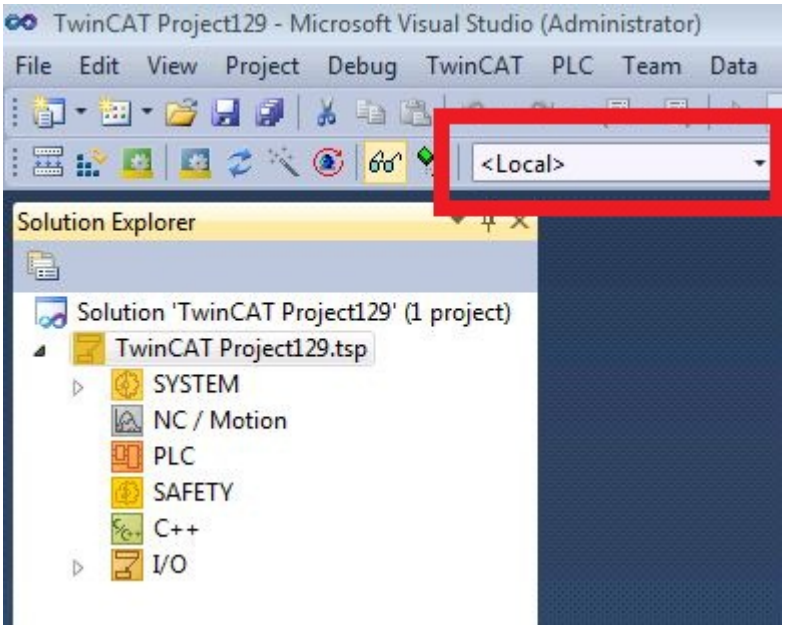
In diesem Artikel wird die Auswahl der Ziel- TwinCAT 3-Laufzeit beschrieben. Das kann sein

- die lokale TwinCAT 3-XAR Laufzeit
- eine TwinCAT 3-XAR Remote-Laufzeit im Netzwerk

Verbindung mit lokaler TwinCAT XAR Laufzeit

Nach Anlage eines neuen TwinCAT 3 Projekts ist standardmäßig das lokale XAR Laufzeitsystem ausgewählt.

1. Wenn nicht, <Local> wählen, um die C++ Module in das lokale TwinCAT 3 Laufzeitsystem herunterzuladen.

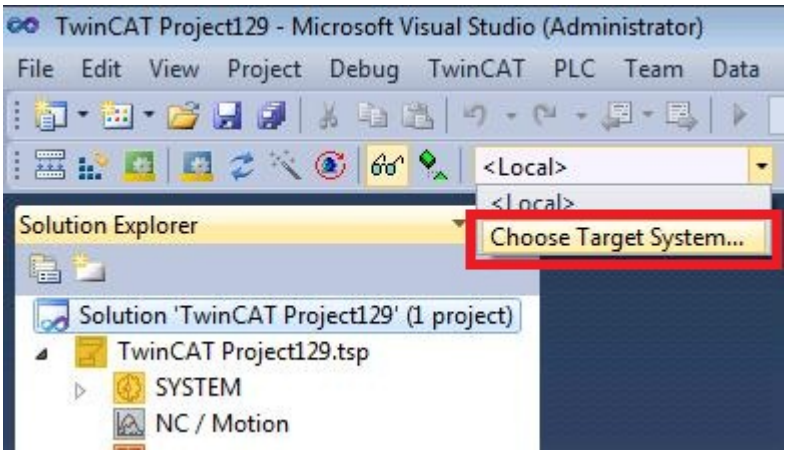


Mit einer TwinCAT 3-XAR Remote-Laufzeit im Netzwerk verbinden.

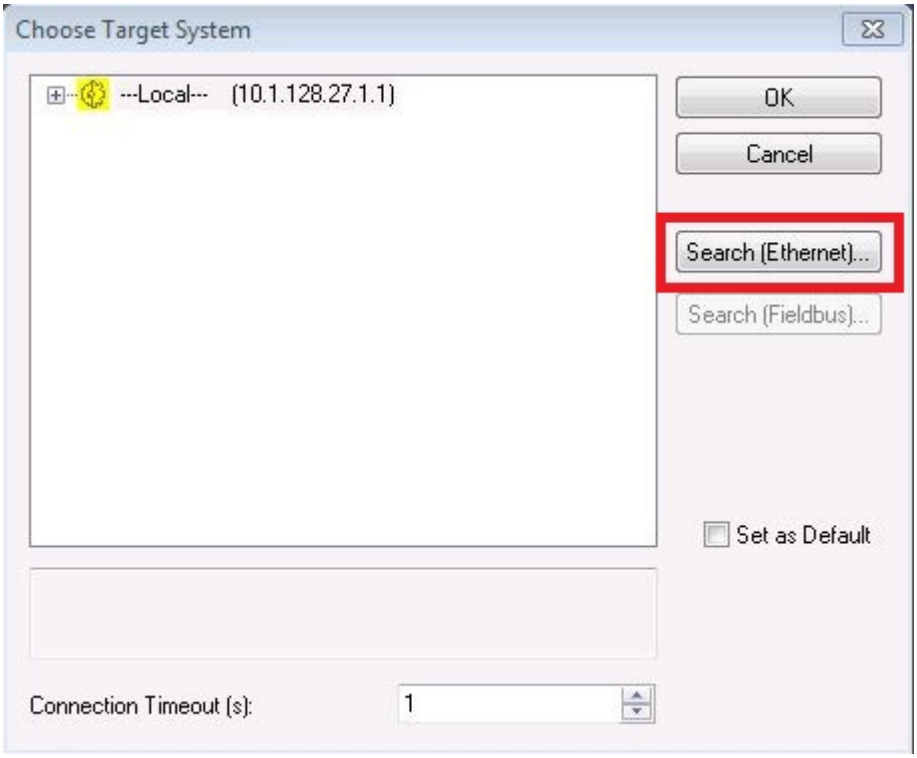
Um eine TwinCAT 3 Solution auf eine TwinCAT 3 XAR Laufzeit auf einem anderen Computer über das Netzwerk herunterzuladen, auszuführen, zu überwachen und zu debuggen, folgen Sie diesen Schritten:

✓ Das TwinCAT Symbol muss grün (läuft) oder blau (Konfigurationsmodus) sein, nicht rot (gestoppt). Wenn TwinCAT gestoppt wurde, funktioniert das Durchsuchen nicht.

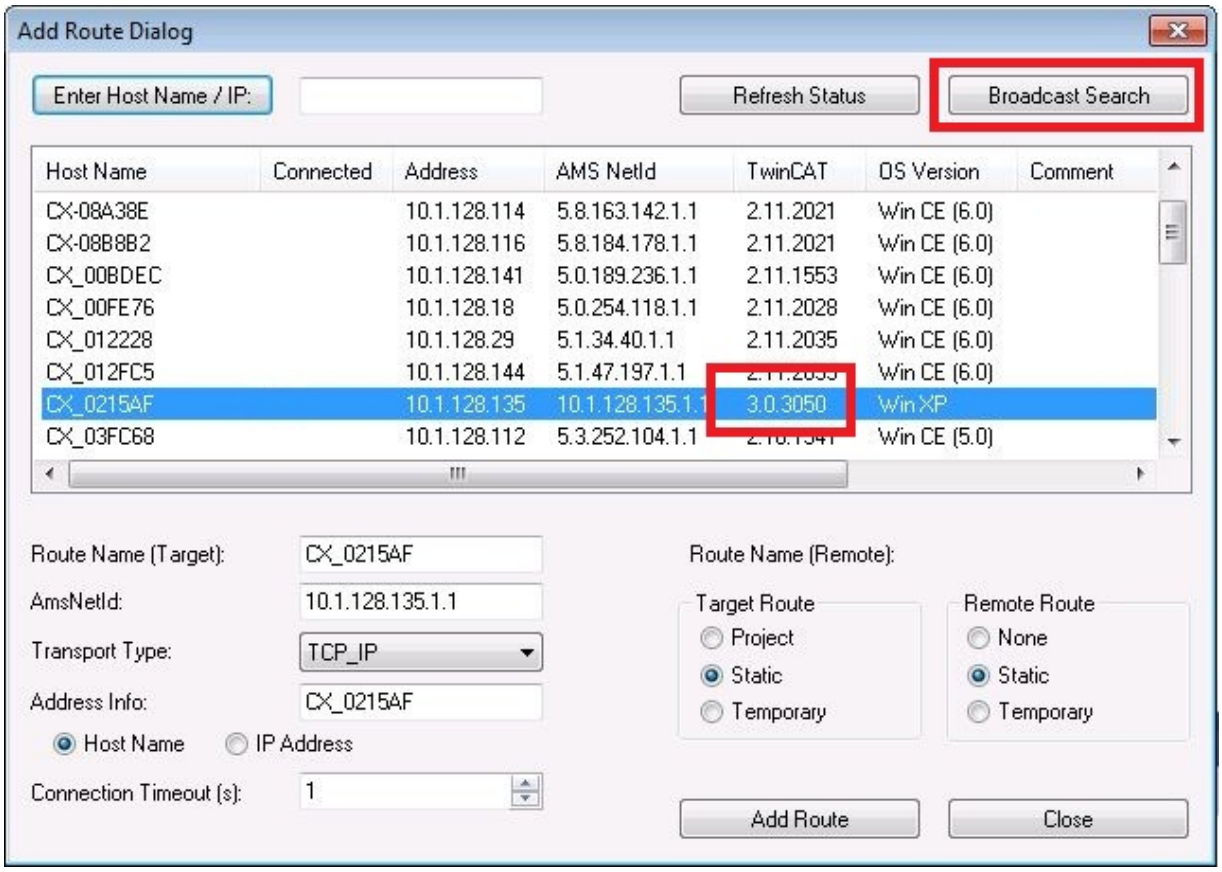
2. Um das Netzwerk nach TwinCAT 3 XAR Laufzeitzielen zu durchsuchen, wählen Sie **„Choose Target System...“**


TwinCAT Project129 - Microsoft Visual Studio (Administrator)
File Edit View Project Debug TwinCAT PLC Team Data
<Local>
Solution Explorer
Choose Target System...
Solution 'TwinCAT Project129' (1 project)
TwinCAT Project129.tsp
SYSTEM
NC / Motion

3. Wählen Sie in diesem Dialog „Search (Ethernet)...“



4. Im nächsten Dialog kann das Netzwerk nach TwinCAT Laufzeitsystemen durchsucht werden.



5. auf **„Broadcast Search“** klicken

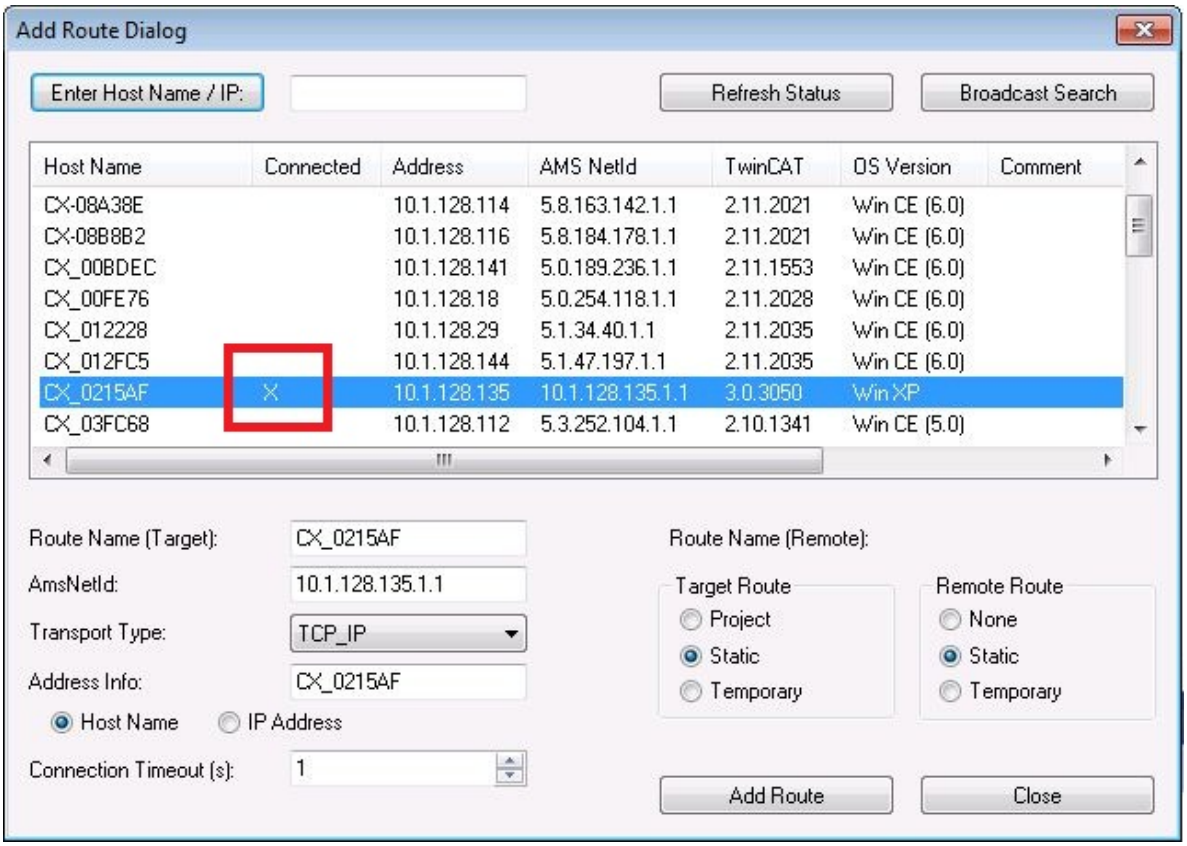
   ⇨ Es erscheinen alle verfügbaren TwinCAT Laufzeitgeräte im Netzwerk.

6. Um ein C++ Programm herunterzuladen und zu überwachen muss die Ziellaufzeit eine TwinCAT 3 3.xxx Laufzeit sein.

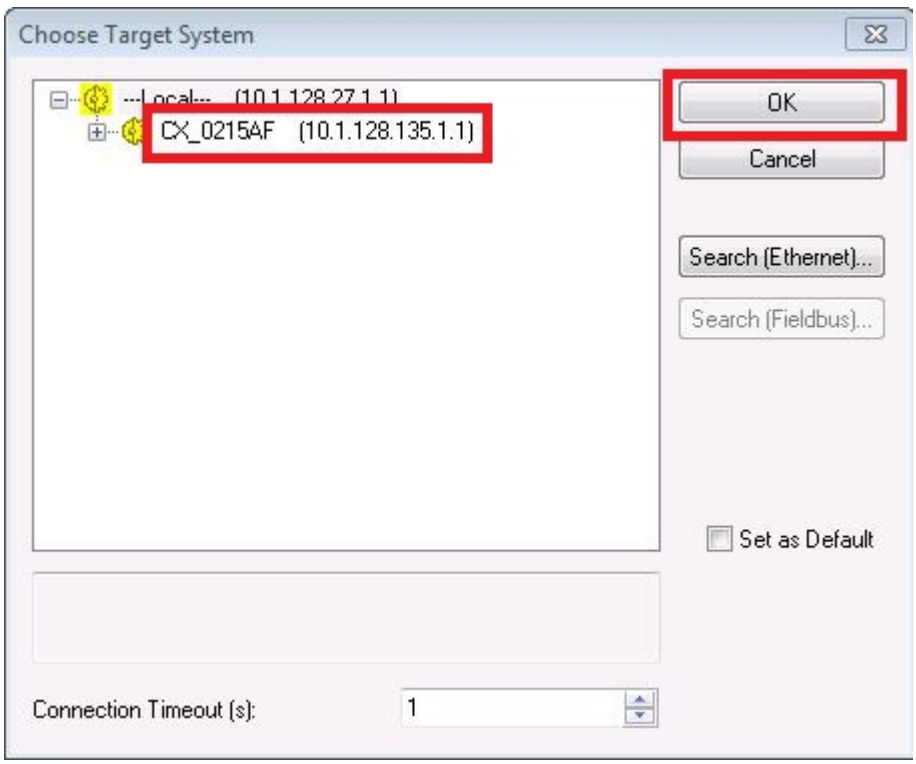
7. Nach Auswahl des korrekten Zielgeräts **„Add Route“** anklicken, um eine ADS-Verbindung zwischen Engineering PC und Laufzeitgeräten herzustellen.
   Alternativ können Sie auch Name oder IP des Zielgeräts direkt in das Feld eingeben und auf **„Enter Host Name / IP“** klicken.

8. Geben Sie aus Sicherheitsgründen Nutzername und Passwort des Ziel-PCs ein.
   Nur gültige Zugangsdaten ermöglichen die Konfiguration einer ADS Route.

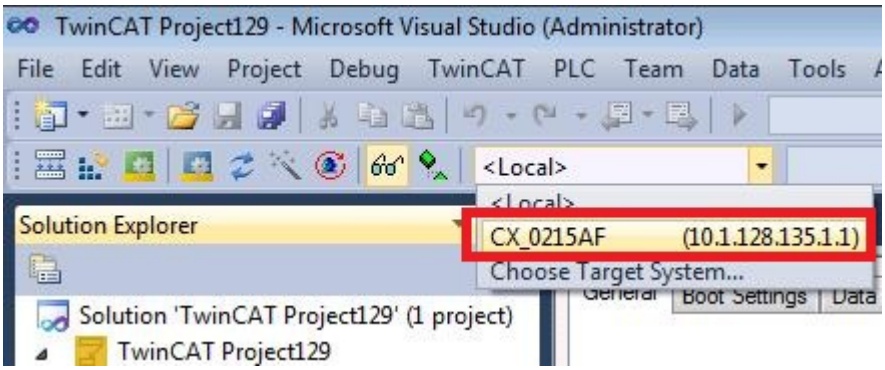
⇨ Wenn Nutzername/Passwort vom Remote-PC akzeptiert werden, ist das gewählte Ziel mit einem „X“ gekennzeichnet, um eine erfolgreich konfigurierte ADS Route anzuzeigen.



9. Wenn das System mit einer Zeitüberschreitung antwortet, erhöhen Sie die Zeit für Verbindungstimeout und versuchen es noch ein Mal.
10. Wenn die Verbindung hergestellt ist, „Close“ wählen, um den Schritt abzuschließen.
    ⇨ Jetzt wird das neue Remote-Gerät aufgelistet.
11. Wählen Sie es als Zielgerät und verlassen den Dialog mit **„OK“**.



⇨ Nach erfolgreicher Konfiguration einer neuen ADS Route kann das neue Zielgerät als Standardgerät ausgewählt werden.


TwinCAT Project129 - Microsoft Visual Studio (Administrator)
File Edit View Project Debug TwinCAT PLC Team Data Tools
<Local>
Solution Explorer
CX_0215AF (10.1.128.135.1.1)
Choose Target System...
Solution 'TwinCAT Project129' (1 project)
TwinCAT Project129
Boot Settings

13.2 Verwenden Sie das Automation Interface

Das Automation Interface kann für den Zugriff auf, die Erstellung von und den Umgang mit TcCOM Modulen verwendet werden.

Von dort aus können gewöhnliche Aufgaben des Systemmanagers wie Verknüpfung von Variablen ausgeführt werden.

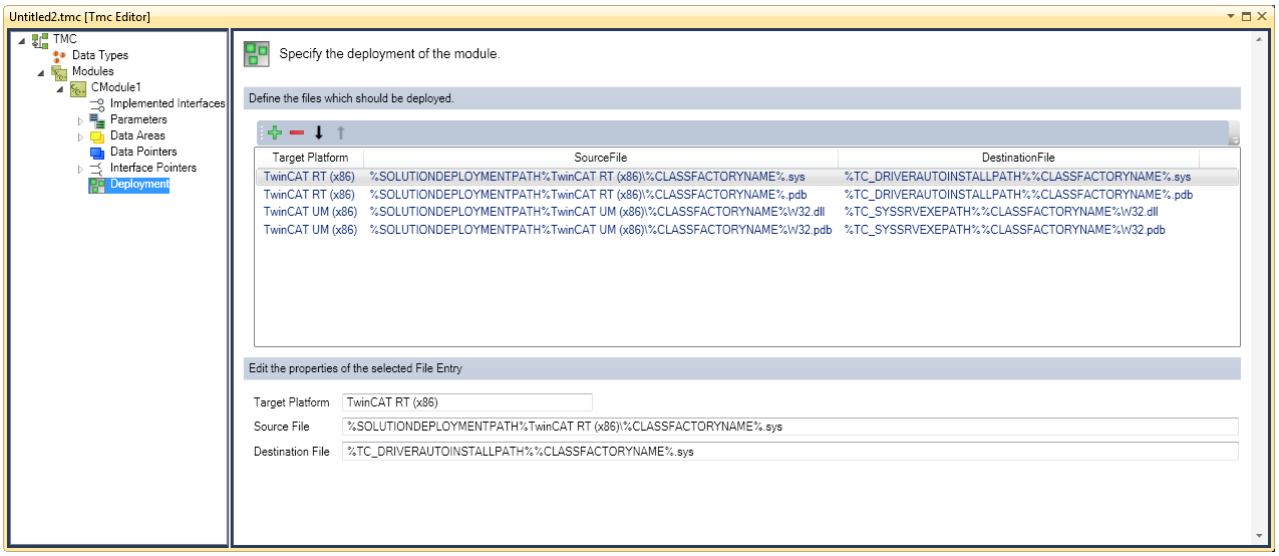
13.3 Bereitstellung

Die Übertragung des kompilierten Treibers auf das Zielsystem wird Bereitstellung (deployment) genannt.

In diesem Artikel wird die Konfiguration der Bereitstellung von TwinCAT 3 Modulen, welche vom TMC Editor unterstützt wird, beschrieben.

Gehen Sie zum TMC Editor

Es muss ein Eintrag für jede unterstützte Plattform, z.B. „TwinCAT RT (x86)“, hinzugefügt werden.



Quell- und Zielpfade für die Bereitstellung können virtuelle Umgebungsvariablen beinhalten, die vom TwinCAT XAE / XAR System aufgelöst werden.

Die nachfolgende Tabelle enthält die Liste dieser unterstützten virtuellen Umgebungsvariablen.

Virtuelle Umgebungsvariable	Registrierungswert (REG_SZ) unter Taste	Defaultwert
	\HKLM\Software\Beckhoff\TwinCAT3	
%TC_INSTALLPATH%	InstallDir	C:\TwinCAT\3.**x** \
%TC_CONFIGPATH%	ConfigDir	C:\TwinCAT\3.**x** \Config\
%TC_TARGETPATH%	TargetDir	C:\TwinCAT\3.**x** \Target\
%TC_SYSTEMPATH%	SystemDir	C:\TwinCAT\3.**x** \System\
%TC_BOOTPRJPATH%	BootDir	C:\TwinCAT\3.**x** \Boot\
%TC_RESOURCEPATH%	ResourceDir	C:\TwinCAT\3.**x** \Target\Resource\
%TC_REPOSITORYPATH%	RepositoryDir	C:\TwinCAT\3.**x** \Repository\
%TC_DRIVERPATH%	DriverDir	C:\TwinCAT\3.**x** \Driver\
%TC_DRIVERAUTOINSTALLPATH%	DriverAutoInstallDir	C:\TwinCAT\3.**x** \Driver\AutoInstall\
%TC_SYSSRVEXEPATH%		C:\TwinCAT\3.**x** \SDK\Bin \TwinCAT UM (x86)\
%CLASSFACTORYNAME%		<Name der Class Factory>

(wobei **„x“** durch die installierte TwinCAT-Version ersetzt wird)

13.4 Module veröffentlichen

Der Abschnitt Module exportieren [▶ 31] beschreibt wie TwinCAT-Module veröffentlicht werden, sodass sie auf ein beliebiges TwinCAT System übertragen und importiert [▶ 32] werden können.

Das Engineering System (XAE) muss sich nicht notwendigerweise auf den gleichen Plattformtyp wie das Ausführungssystem befinden. Hierzu erstellt TwinCAT im Verlauf der Veröffentlichung alle Versionen des Moduls.

Bestimmte Anwendungsfälle erfordern eine Anpassung der Veröffentlichung der Module:

- Beim Arbeiten in einer reinen 32Bit (x86) Umgebung können die x64 Erstellungen übersprungen werden, sodass keine Zertifikate benötigt werden.
- Die User Mode (UM) Erstellungen können übersprungen werden, wenn sie nicht verwendet werden.

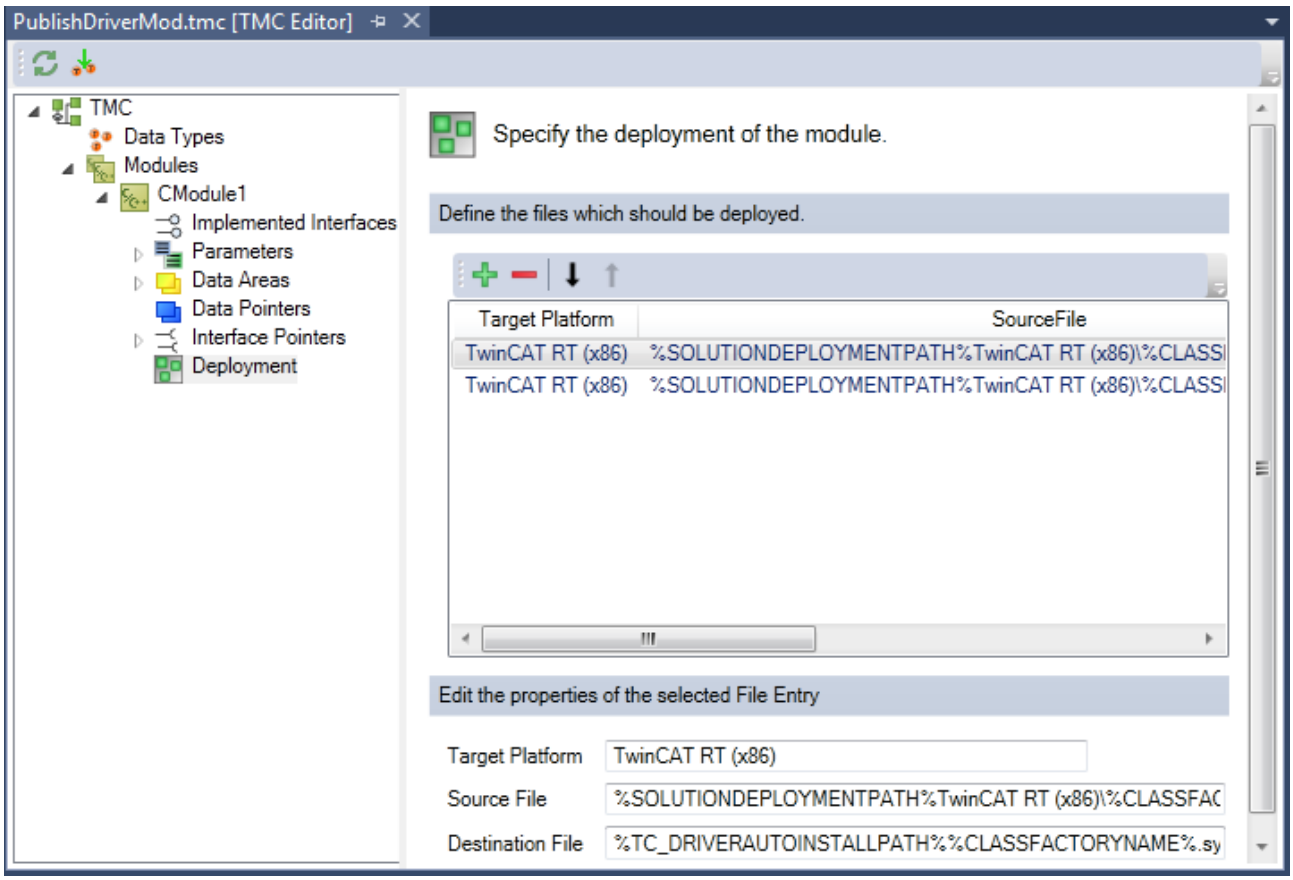
Hinweis

Migration

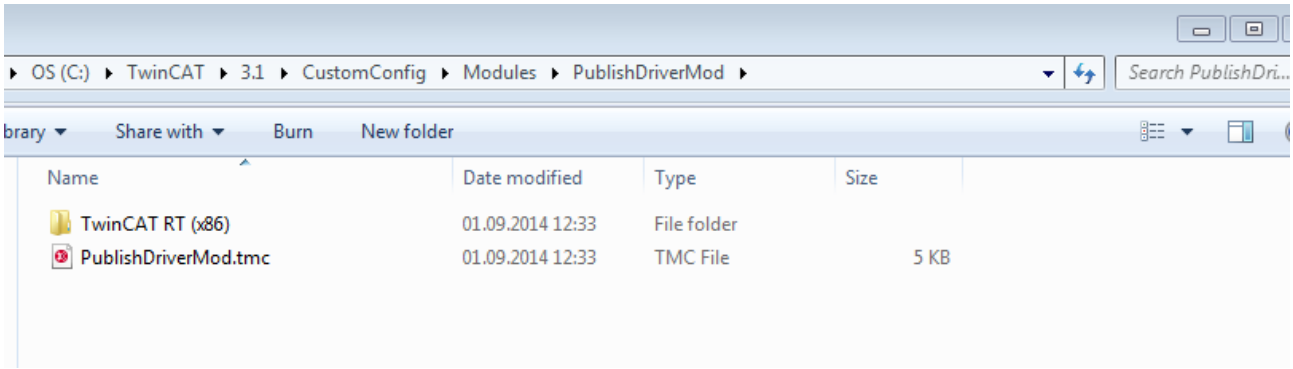
Wenn ein Build nicht im veröffentlichten Modul enthalten ist, kann die entsprechende Plattform nicht als Ausführungssystem verwendet werden.

Bitte die entsprechenden Zielplattformen in Bereitstellung [▶ 111] des TMC Editors hinzufügen / entfernen.

Folgende Bereitstellungskonfiguration als Beispiel:



stellt ausschließlich die TwinCAT RT (x86) Build zur Verfügung:



13.5 Clone

Die Laufzeitdateien können mittels Dateikopie von einer zur anderen Maschine übertragen werden, wenn diese von derselben Plattform stammen und mit äquivalenter Hardware-Ausrüstung verbunden sind.

Die folgenden Schritte beschreiben ein einfaches Verfahren zum Übertragen einer Binärkonfiguration von einer Maschine „Quelle“ zu einer anderen Maschine „Ziel“.

✓ Bitte den Ordner C:\TwinCAT\3.x\Boot auf der Quell-Maschine leeren.

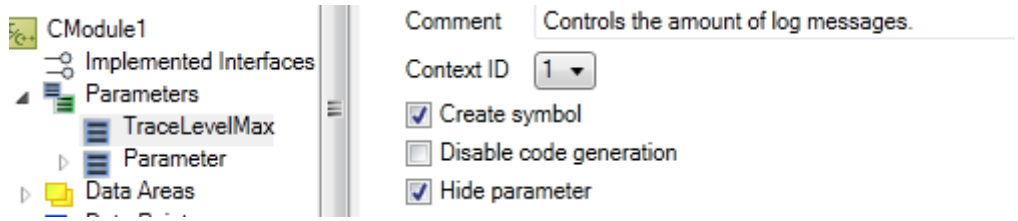
1. Das Modul auf der Quellmaschine erstellen (oder aktivieren).
2. Den Ordner C:\TwinCAT\3.x\Boot von der Quelle zum Ziel übertragen.
3. Den Treiber selber von C:\TwinCAT\3.x\Driver\AutoInstall**MYDRIVER.sys** und **MYDRIVER.pdb** übertragen.

⇨ Die Zielmaschine kann gestartet werden.

Hinweis	**Umgang mit Lizenzen** Beachten Sie, dass die Lizenzen nicht auf diese Weise übertragen werden können. Verwenden Sie bitte vorinstallierte Lizenzen, Volumenlizenzen oder andere Mechanismen, um Lizenzen bereitzustellen.

13.6 Zugriff auf Variablen über ADS

Variablen von C++ Modulen sind über ADS erreichbar, wenn die Variable im TMC Editor als „Create Symbol“ gekennzeichnet sind:



Der Name der Variablen für den Zugriff per ADS ist abgeleitet vom Name der Instanz.

Für den TraceLevelMax Parameter könnte er lauten:

```
Untitled1_Obj1 (CModule1).TraceLevelMax
```

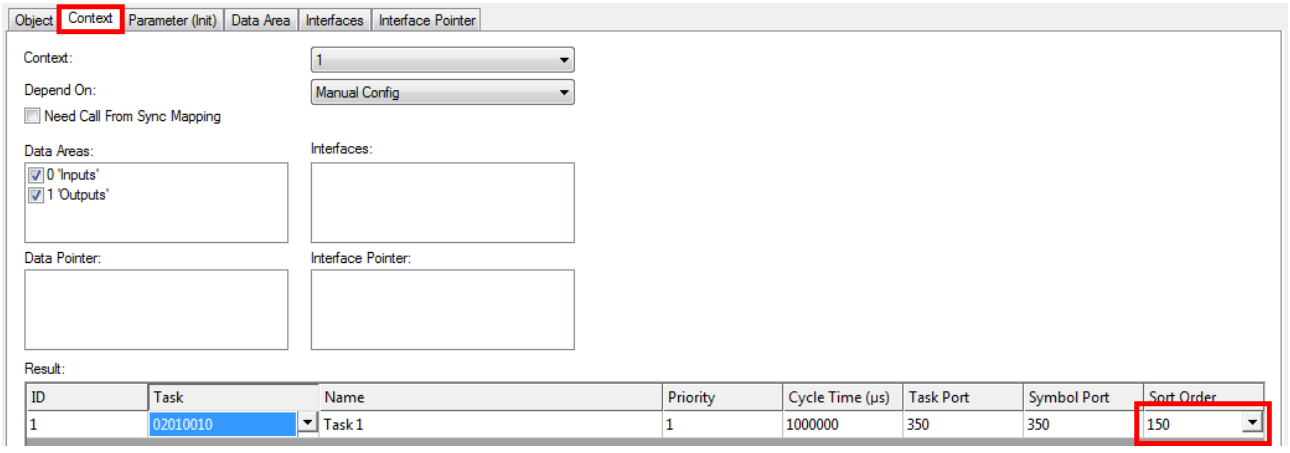
13.7 TcCallAfterOutputUpdate für C++ Module

Vergleichbar mit dem SPS Attribut TcCallAfterOutputUpdate können C++ Module nach der Ausgang-Aktualisierung aufgerufen werden.
Äquivalent zu ITcCyclic [▶ 130] Schnittstelle: Verwenden Sie bitte die ITcPostCyclic [▶ 143] Schnittstelle

13.8 Reihenfolgebestimmung der Ausführung in einem Task

Es können verschiedene Modulinstanzen einem Task zugewiesen werden, sodass der Benutzer einen Mechanismus benötigt, um die Reihenfolge der Ausführung im Task festzulegen.

Das Schlüsselwort lautet „Sort Order“, das im Kontext [▶ 114] des TwinCAT Module Instance Configurator [▶ 112]konfiguriert wird.



Siehe Beispiel26: Ausführungsreihenfolge in einem Task [▶ 254], wie das zu implementieren ist.

13.9 Stack Size > 4kB verwenden

Die standardmäßige Stack-Größe ist auf 4KB begrenzt. Wird diese Grenze überschritten, dann tritt die Fehlermeldung

```
Error LNK2001: unresolved external symbol __chkstk
```

auf.

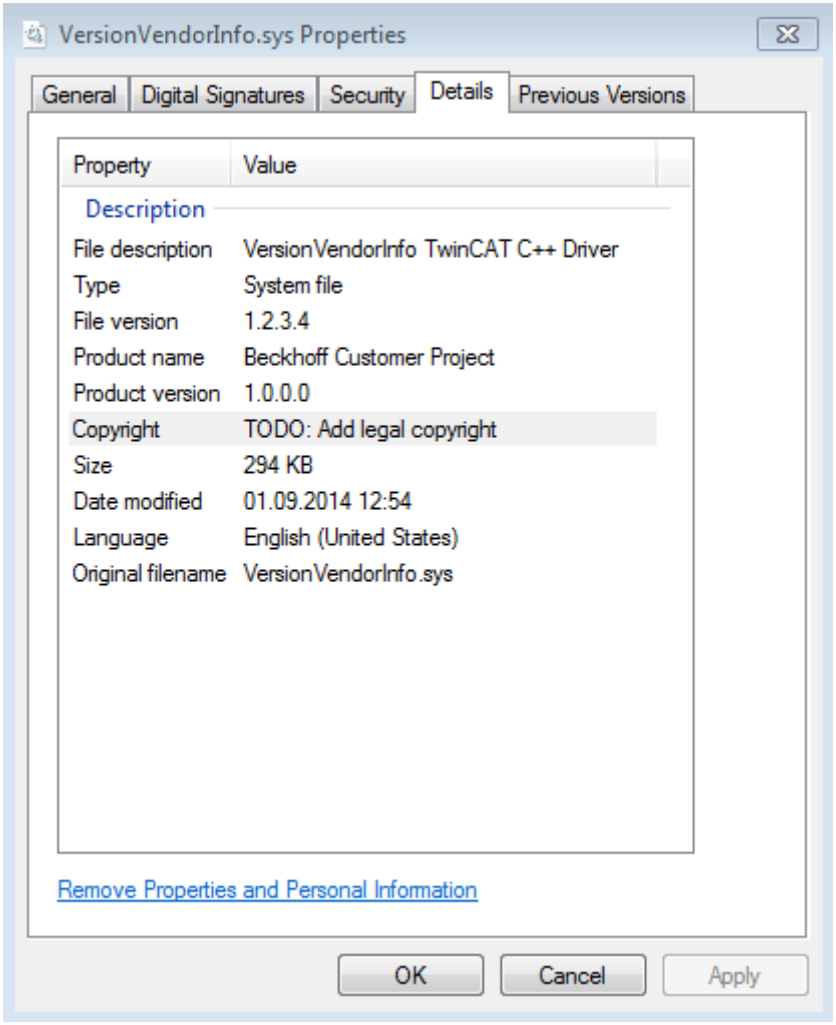
Man kann die von TwinCAT bereit gestellte Bibliothek „chkstk.lib“ verwenden, indem man dem Projekt die Eigenschaftendatei „TwinCAT RT ChkStk“ (standardmäßig in c:\ C:\TwinCAT\3.**x**\sdk\VsProps\) hinzufügt.

HINWEIS! Aber es sind auch Stack-Größen über 4KB möglich: Wir empfehlen Speicherplatz über Modulsymbole zu allozieren, die also im TMC Editor [▶ 69]definiert sind.

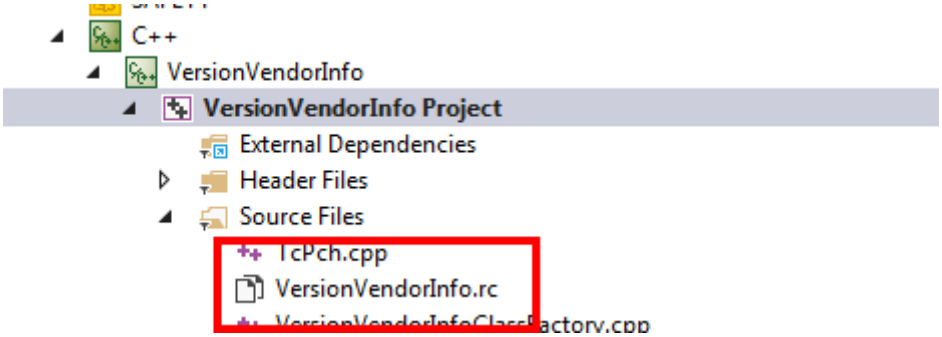
13.10 Siehe Version/Herstellerinformation

Windows bietet einen Mechanismus, um Hersteller- und Versionsressourcen abzufragen, die im Verlauf einer .rc Datei für die Kompilationszeit definiert sind.

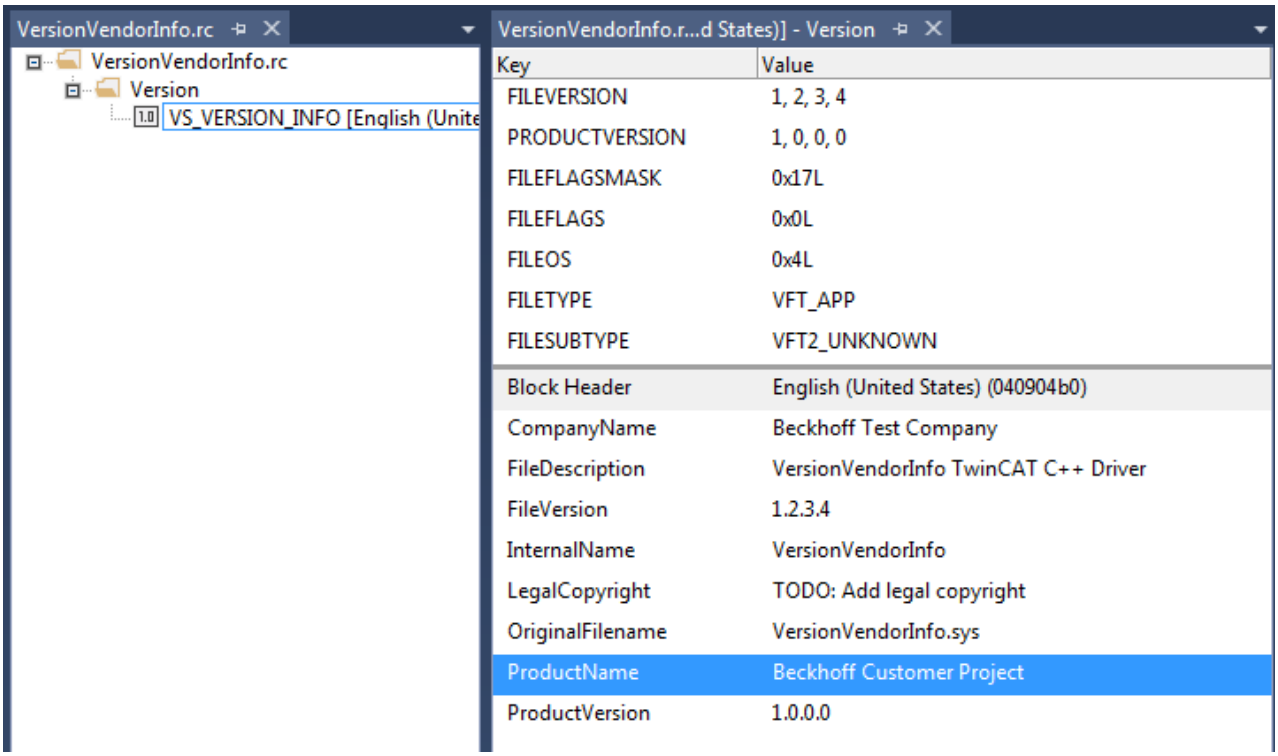
Diese sind z.B. über den Karteireiter Details von jeder Eigenschaften-Datei zugänglich.



TwinCAT bietet dieses Verhalten über die bekannten Windows Mechanismen von .rc Dateien, die im Verlauf der TwinCAT C++ Projekterstellung erzeugt werden.



Bearbeiten Sie bitte die .rc-Datei im Source Files Ordner mit dem Resource Editor, um diese Eigenschaften festzulegen:

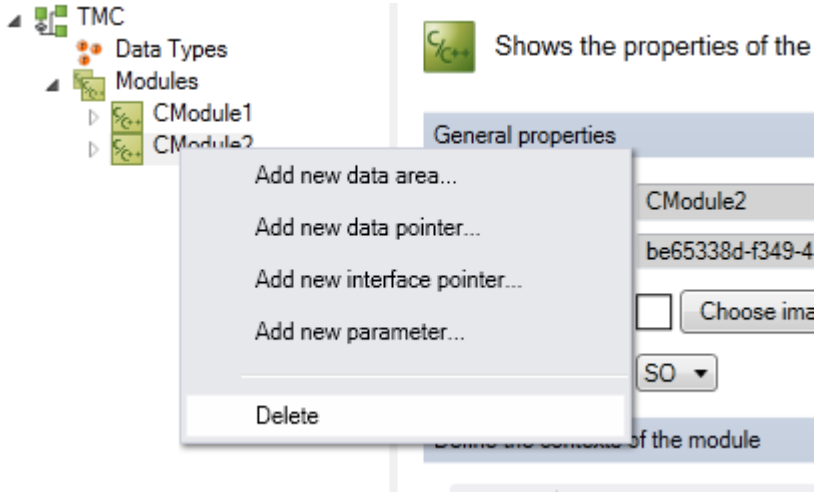


13.11 Modul löschen

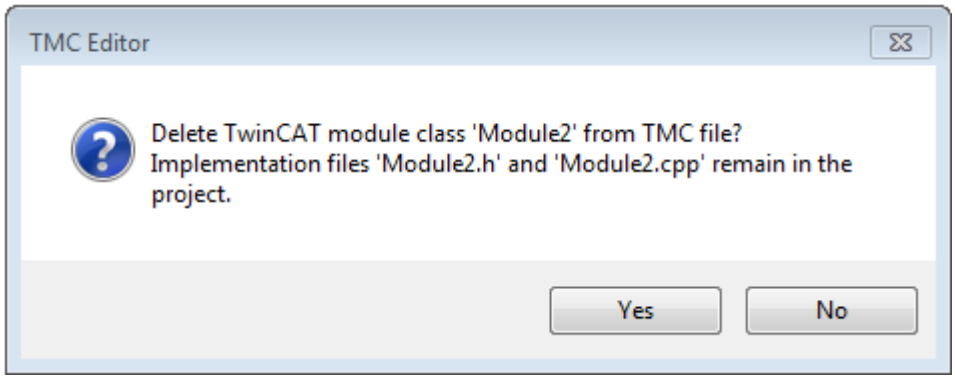
Ein TwinCAT C++ Modul kann mit Hilfe des TMC Editors aus einem C++ Projekt gelöscht werden.

1. Rechtsklicken Sie auf das Modul (hier: CModule2)

2. „Delete“ auswählen



3. Löschung über TMC bestätigen


TMC Editor
Delete TwinCAT module class 'Module2' from TMC file?
Implementation files 'Module2.h' and 'Module2.cpp' remain in the project.
Yes
No

4. Beachten Sie, dass die .cpp und .h Dateien erhalten bleiben – gegebenenfalls diese manuell löschen. Löschen anderer betreffender Komponenten (z.B. Header- Dateien, Strukturen). Siehe Compiler-Fehlermeldungen für weitere Auskünfte.

13.12 Initialisierung von TMC-Membervariablen

Alle Membervariablen eines TcCOM Moduls müssen initialisiert werden. Der TMC Code Generator unterstützt dies mit

```
///<AutoGeneratedContent id="MemberInitialization">
```

Wird durch den TMC Code Generator ersetzt durch:

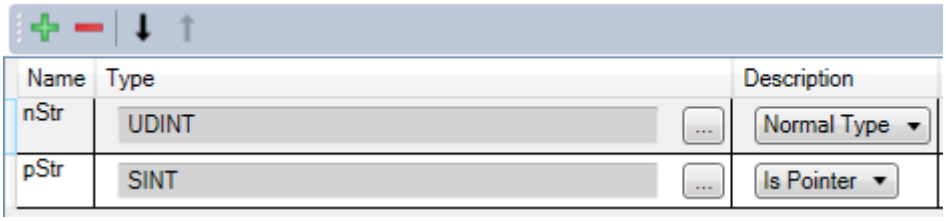
```
///<AutoGeneratedContent id="MemberInitialization">
m_TraceLevelMax = tlAlways;
memset(&m_Parameter, 0, sizeof(m_Parameter));
memset(&m_Inputs, 0, sizeof(m_Inputs));
memset(&m_Outputs, 0, sizeof(m_Outputs));
///</AutoGeneratedContent>
```

Die mit dem TwinCAT C++ Assistenten vor TwinCAT 3.1 Build 4018 generierten Projekte verwenden diese Eigenschaft nicht, können aber einfach angepasst werden, indem diese Zeile im entsprechenden Code (z.B. Konstruktor) hinzugefügt wird:

```
///<AutoGeneratedContent id="MemberInitialization">
```

13.13 SPS-Zeichenketten als Methodenparameter verwenden

Um eine Zeichenkette von SPS nach C++ als Methodenparameter zu übergeben, verwenden Sie bitte einen Zeiger mit Längeninformation beim Deklarieren der Methode in TMC:



Eine solche Methode kann mittels Implementierung einer Methode innerhalb des Wrapper-Funktionsbausteins aufgerufen werden:

```
METHOD SetString : HRESULT
VAR_INPUT
    sSent : STRING(80);
END_VAR
```

```
IF (ipStateMachine <> 0) THEN
    SetString := ipStateMachine.SetString(SIZEOF(sSent),ADR(sSent));
END_IF
```

Der Grund ist der unterschiedliche Umgang mit Methodenparametern in beiden Welten:

- PLC: Verwendet den Werteaufruf (call by value) für STRING(nn) Datentypen
- TwinCAT C++ (TMC): Verwendet Verweisaufruf (call by reference)

13.14 Bibliotheken von Drittanbietern

In Kernel-Modus vorliegender C/C++ Code kann nicht mit Bibliotheken von Drittanbietern, die für die Ausführung im User-Modus konzipiert wurden, verknüpft werden und diese ausführen. Somit besteht keine Möglichkeit, eine beliebige DLL in TwinCAT C++ Modulen zu verwenden.

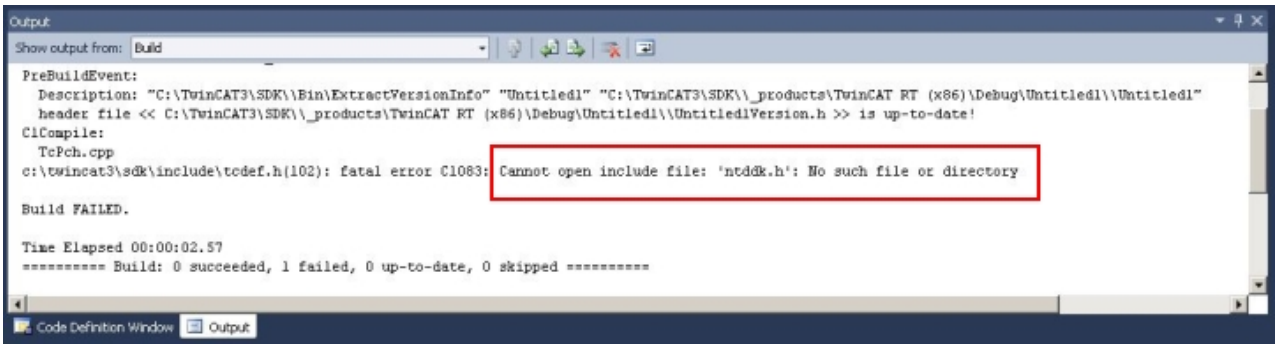
Stattdessen kann die Verbindung zur TwinCAT 3 Echtzeitumgebung über ADS-Kommunikation realisiert werden. Man kann eine User-Modus Anwendung implementieren, die die Bibliothek von Drittanbietern verwendet, welche TwinCAT Funktionalitäten über ADS zur Verfügung stellt.

14 Fehlersuche

Liste von Fallstricken und Pannen beim Umgang mit TwinCAT C++ Modulen.

14.1 Build - „Cannot open include file ntddk.h“

Beim Erstellen eines TwinCAT C++ Projekts zeigt diese Fehlermeldung ein WinDDK-Problem auf Ihrem Engineering Computer.

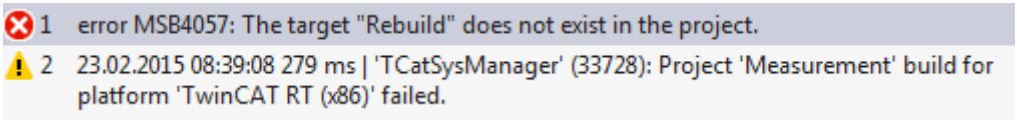


Bei Auftreten dieser Fehlermeldung prüfen Sie nach WinDDK-Installationshandbuch [▶ 13]:

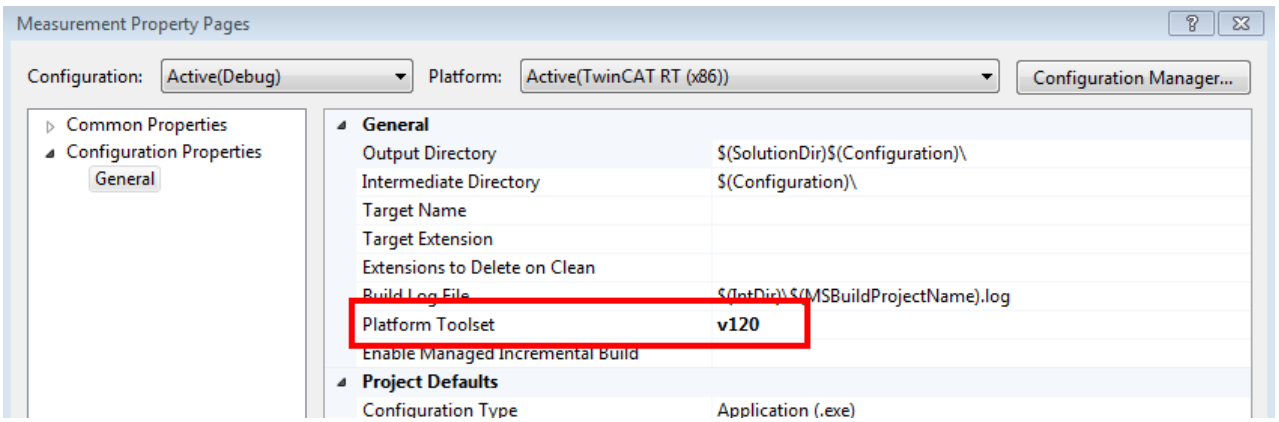
- Vergewissern, dass WinDDK installiert ist.
- Umgebungsvariable „WinDDK7“ und ihr konfigurierter Wert muss existieren, siehe Beschreibung in oben erwähntem Dokument. Der Wert muss dem Pfad entsprechen, wo WinDDK installiert ist, einschließlich des ersten Unterordners. Nach einer Änderung dieses Werts muss der Computer neu gestartet werden.

14.2 Build - „The target ... does not exist in the project“

Insbesondere bei der Übertragung einer TwinCAT Solution von einer Maschine auf die andere gibt Visual Studio möglicherweise Fehlermeldungen aus, gemäß derer alle Ziele (wie Build, Rebuild, Clean) nicht im Projekt existieren.

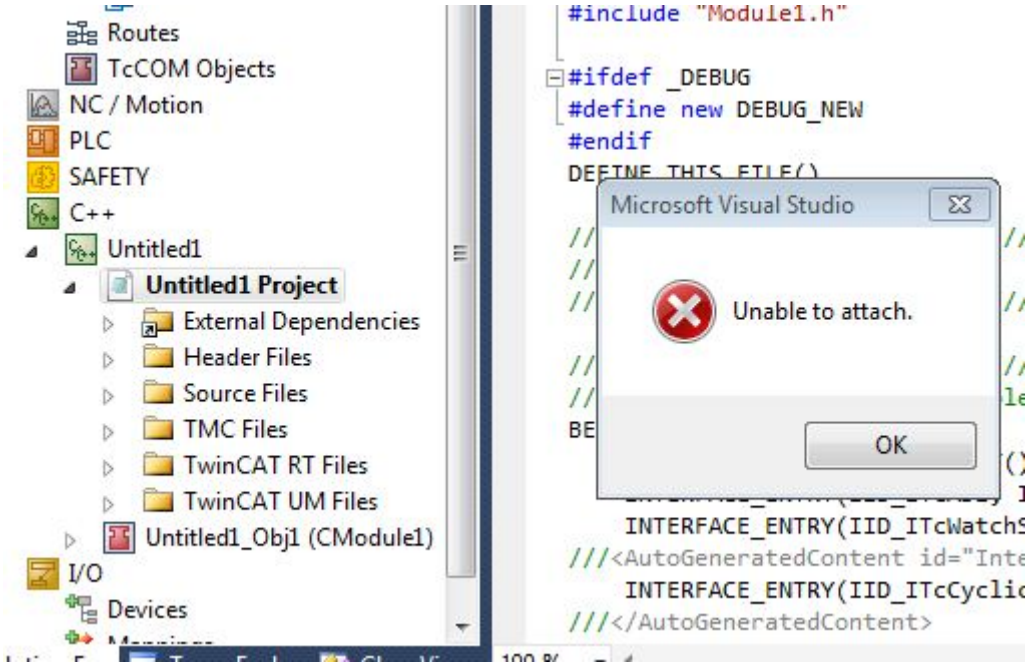


Bitte die Konfiguration von „platform toolset“ des C++ Projekts überprüfen. Muss gegebenenfalls neu konfiguriert werden, wenn Solutions von einer zu anderen Visual Studio Version migrieren:

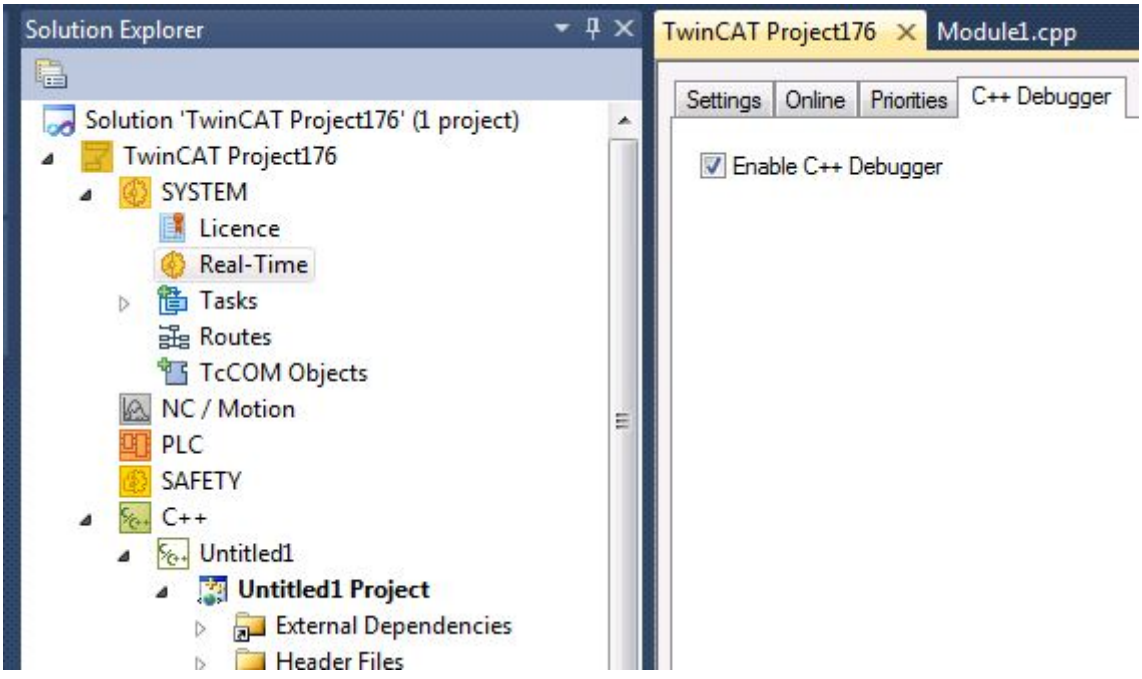


14.3 Debug - „Unable to attach“

Wenn diese Fehlermeldung beim Starten des Debuggers zum Debugging eines TwinCAT C++ Projekts erscheint, fehlt ein Konfigurationsschritt:

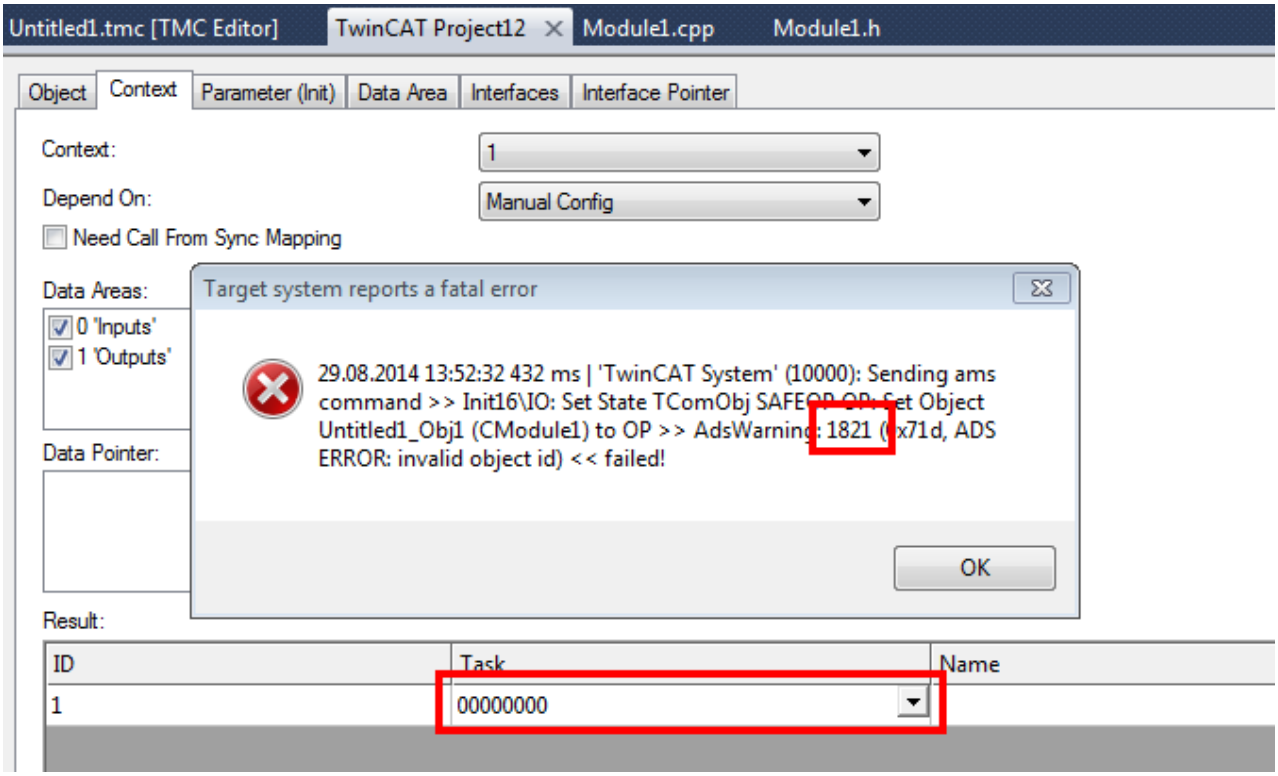


In diesem Fall zu „System -> Real-Time“ navigieren, Registerkarte „C++ Debugger“ wählen und Option „Enable C++ Debugger“ aktivieren.



14.4 Activation – „invalid object id“ (1821/0x71d)

Wenn im Verlauf des Starts der ADS Return Code 1821 / 0x71d berichtet wird, bitte den Kontext der Modulinstanz wie in Schnellstart [▶ 48] beschrieben überprüfen.



14.5 Fehlermeldung – VS2010 und LNK1123/COFF

Während der Kompilation eines TwinCAT C++ Moduls, weist die Fehlermeldung

```
LINK : fatal error LNK1123: failure during conversion to COFF: file invalid or corrupt
```

darauf hin, dass ein Visual Studio 2010 verwendet wird, aber ohne Service Pack 1, der für TwinCAT C++ Module erforderlich [▶ 12] ist.

Bitte das Installationsprogramm für den Service Pack von Microsoft herunterladen.

14.6 Verwendung von C++ Klassen in TwinCAT C++ Modulen

Beim Hinzufügen von (nicht TwinCAT) C++ Klassen bei der Verwendung vom Visual Studio-Kontextmenü Add->Class..., der Compiler / Linker meldet:

```
Error 4 error C1010: unexpected end of file while looking for precompiled
header. Did you forget to add '#include ""' to your source?
```

Fügen Sie bitte die folgenden Zeilen am Anfang Ihrer erzeugten Klassendatei ein:

```
#include "TcPch.h"
#pragma hdrstop
```

TEIL 15

C/C++ Beispiele

15.1 Übersicht ... 196
15.2 Beispiel01: Zyklisches Modul mit IO ... 198
15.3 Beispiel02: Zyklische C++ Logik, die IO vom IO Task verwendet ... 199
15.4 Beispiel03: C++ als ADS Server ... 200
15.4.1 Beispiel03: Der in C++ geschriebene TC3 ADS Server ... 201
15.4.2 Beispiel03: ADS Client UI in C# ... 205
15.5 Beispiel05: C++ CoE Zugriff über ADS ... 209
15.6 Beispiel06: UI-C#-ADS Client lädt die Symbolik vom Modul hoch ... 210
15.7 Beispiel07: Empfang von ADS Notifications ... 215
15.8 Beispiel10: Modulkommunikation: Verwendung von Datenzeigern ... 216
15.9 Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf ... 217
15.9.1 Beispiel11: Eine Methode zur Verfügung stellendes TwinCAT 3 C++ Modul ... 218
15.9.2 Beispiel11: SPS um eine Methode aufzurufen, die von einem anderen Modul angeboten wird (hier TwinCAT 3 C++ Modul) ... 231
15.10 Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an ... 247
15.11 Beispiel12: Modulkommunikation: Verwendet IO Mapping ... 248
15.12 Beispiel19: Synchroner Dateizugriff ... 248
15.13 Beispiel20: FileIO-Write ... 249
15.14 Beispiel20a: FileIO-Cyclic Read / Write ... 250
15.15 Beispiel22: Automation Device Driver (ADD): Zugang DPRAM ... 251
15.16 Beispiel25: Statische Bibliothek ... 252
15.17 Beispiel26: Ausführungsreihenfolge in einem Task ... 254
15.18 Beispiel30: Zeitmessung ... 256
15.19 Beispiel31: Funktionsbaustein TON in TwinCAT3 C++ ... 257
15.20 Beispiel35: Ethernet Zugriff ... 257
15.21 Beispiel37: Daten archivieren ... 259

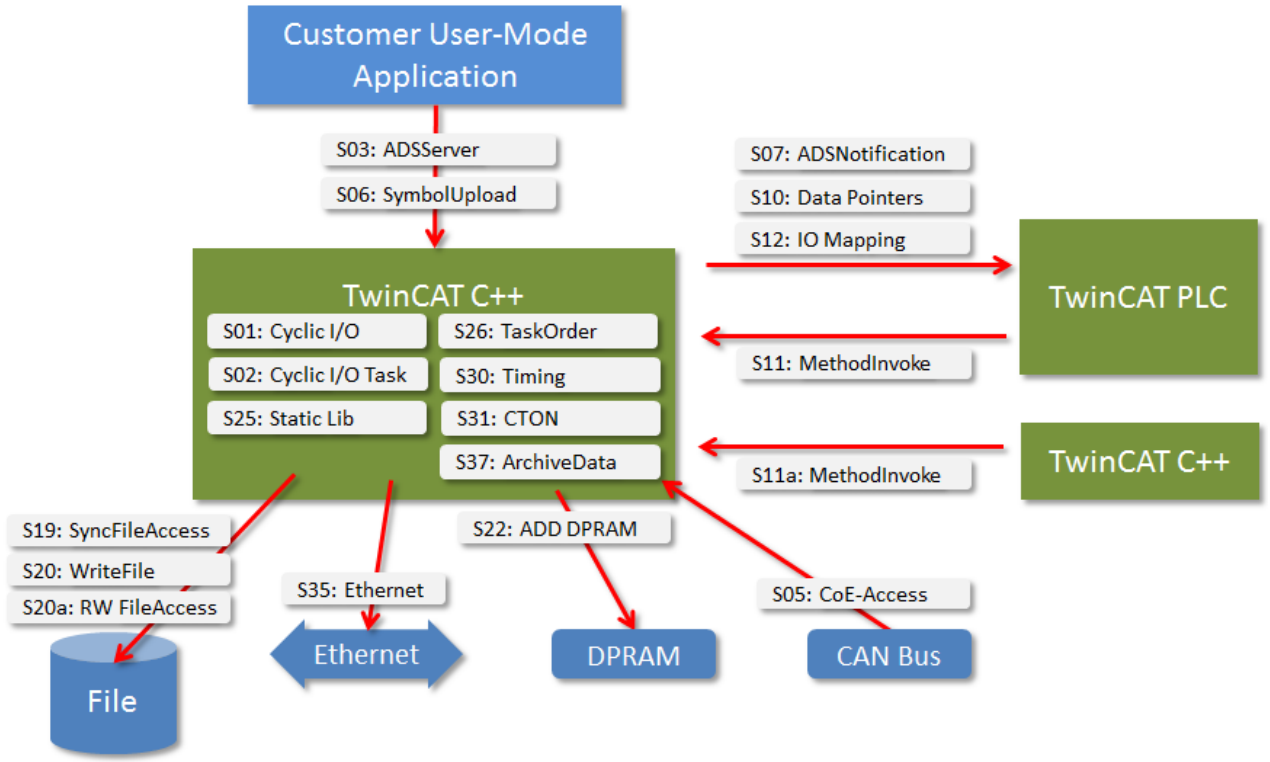
15 C/C++ Beispiele

15.1 Übersicht

Übersicht

Es stehen zahlreiche Beispiele zur Verfügung - weitere Beispiele werden folgen.

Dieses Bild zeigt einen Überblick in grafischer Form und legt dabei den Schwerpunkt auf die Interaktionsmöglichkeiten eines C++ Moduls.



Darüber hinaus ist dies eine Tabelle mit kurzen Beschreibungen der Beispiele.

Number	Beschreibung	Beschreibung
01	Beispiel01: Zyklisch mit IO-Modul [▶ 198]	Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, das ein mit physikalischem IO gemapptes IO-Modul verwendet. Dieses Beispiel beschreibt den Schnellstart zwecks Erstellung eines C++ Moduls, das einen Zähler bei jedem Zyklus inkrementiert und den Zähler dem logischen Ausgang „Value“ im Datenbereich zuweist. Der Datenbereich kann dem physikalischen IO oder einem anderen logischen Eingang einer anderen Modulinstanz zugewiesen werden.
02	Beispiel02: Zyklisch mit IO Task [▶ 199]	Dieses Beispiel beschreibt die Flexibilität von C++ Code bei der Arbeit mit IOs, welche an dem Task konfiguriert sind. Dank dieser Herangehensweise kann ein abschließend kompiliertes C++ Modul weit flexibler auf verschiedene, mit dem IO Task verbundene IOs einwirken. Eine Anwendung könnte darin bestehen, zyklische analoge Eingangskanäle zu überprüfen, wobei die Anzahl Eingangskanäle von einem Projekt zum anderen unterschiedlich sein kann.
03	Beispiel03: ADS Server Client [▶ 200]	Beschreibt den Entwurf und die Implementierung einer eigenen ADS-Schnittstelle in einem C++ Modul. Das Beispiel enthält zwei Teile: - in TC3-C++ implementierter ADS Server mit benutzerspezifischer ADS-Schnittstelle - in C# implementierte ADS Client UI, die benutzerspezifische ADS-Meldungen an den ADS Server sendet
04		
05	Beispiel05: CoE Zugriff über ADS [▶ 209]	Zeigt wie über ADS auf CoE Register von EtherCAT-Geräten zugegriffen werden kann
06	Beispiel06: ADS C#-Client lädt ADS-Symbole hoch [▶ 210]	Zeigt, wie über die ADS-Schnittstelle auf Symbole in einem ADS Server zugegriffen werden kann. C# ADS Client tritt in Verbindung mit einem in SPS/ C++ / Matlab implementierten Modul, Hochladen der verfügbaren Symbolinformation und Lese-/Schreiben-Abonnieren für Prozesswerte.
07	Beispiel07: Empfang von ADS Notifications [▶ 215]	Beschreibt die Implementierung eines TC3 C++ Moduls, das ADS Notifications bezüglich Datenänderungen auf anderen Modulen empfängt.
10	Beispiel10: Modulkommunikation: Verwendung von Datenzeigern [▶ 216]	Beschreibt die Interaktion zwischen zwei C++ Modulen mit einem direkten Zeiger (DataPointer). Die beiden Module müssen auf demselben CPU-Kern im selben Echtzeitkontext ausgeführt werden.
11	Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf [▶ 217]	Dieses Beispiel beinhaltet zwei Teile • Ein C++ Modul, das als Zustandsmaschine fungiert, die eine Schnittstelle mit Methoden zum Starten/ Stoppen, aber auch zum Setzen/Erhalten der Zustandsmaschine zur Verfügung stellt. • Zweites SPS-Modul um mit erstem Modul zu interagieren, indem Methoden vom C++ Modul aufgerufen werden

11a	Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an [▶ 247]	Dieses Beispiel beinhaltet zwei Klassen in einem Treiber (kann auch zwischen zwei Treibern gemacht werden) • Ein Modul, das eine Berechnungsmethode bereitstellt Der Zugriff ist durch eine Critical Section geschützt • zweites Modul, das als Aufrufer agiert, um die Methode im anderen Modul zu verwenden
12	Beispiel12: Modulkommunikation: Verwendet IO Mapping [▶ 248]	Beschreibt wie zwei Module über das Mapping von Symbolen des Datenbereichs verschiedener Module unter einander interagieren können. Die beiden Module können auf demselben oder auf verschiedenen CPU-Kernen ausgeführt werden.
19	Beispiel19: Synchroner Dateizugriff [▶ 248]	Beschreibt, wie die File-IO-Funktionalität bei C++ Modul auf synchrone Art und Weise verwendet werden kann. Das Beispiel schreibt Prozesswerte in eine Datei. Das Beschreiben der Datei wird von einem deterministischen Zyklus veranlasst - die Ausführung von File IO ist entkoppelt (asynchron), d.h.: Der deterministische Zyklus läuft weiter und wird nicht durch das Schreiben in der Datei behindert. Der Status der Routine für entkoppeltes Schreiben in der Datei kann überprüft werden.
20	Beispiel20: FileIO-Write [▶ 249]	Beschreibt, wie die File-IO-Funktionalität bei C++ Modul verwendet werden kann. Das Beispiel schreibt Prozesswerte in eine Datei. Das Beschreiben der Datei wird von einem deterministischen Zyklus veranlasst - die Ausführung von File IO ist entkoppelt (asynchron), d.h.: Der deterministische Zyklus läuft weiter und wird nicht durch das Schreiben in der Datei behindert. Der Status der Routine für entkoppeltes Schreiben in der Datei kann überprüft werden.
20a	Beispiel20a: FileIO-Cyclic Read / Write [▶ 250]	Ist ein umfangreicheres Beispiel als S20 und S19. Es beschreibt zyklischen Lese- und/oder Schreibzugriff auf Dateien von einem TC3-C++ Modul aus.
22	Beispiel22: Automation Device Driver (ADD): Zugang DPRAM [▶ 251]	Beschreibt, wie der TwinCAT Automation Device Driver (ADD) für den Zugriff auf die DPRAM zu schreiben ist.
25	Beispiel25: Statische Bibliothek [▶ 252]	Beschreibt, wie die in einem anderen TC3 C++ Modul enthaltene TC3 C++ statische Bibliothek verwendet werden kann.
26	Beispiel26: Ausführungsreihenfolge in einem Task [▶ 254]	Dieser Artikel beschreibt die Bestimmung der Taskausführungsreihenfolge, wenn einem Task mehr als ein Modul zugeordnet ist.
30	Beispiel30: Zeitmessung [▶ 256]	Beschreibt die Messung der TC3 C++ Zyklus- oder Ausführungszeit.
31	Beispiel31: Funktionsbaustein TON in TwinCAT3 C++ [▶ 257]	Dieser Artikel beschreibt die Implementierung eines Verhaltens in C++, das vergleichbar mit einem TON Funktionsbaustein von SPS / 61131 ist.
37	Beispiel37: Daten archivieren [▶ 259]	Beschreibt das Laden und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung.

15.2 Beispiel01: Zyklisches Modul mit IO

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, dessen Modul-IO mit einem physikalischen IO gemappt ist.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Dieses Beispiel beschreibt den Schnellstart zwecks Erstellung eines C++ Moduls, das einen Zähler bei jedem Zyklus inkrementiert und den Zähler dem logischen Ausgang „Value“ im Datenbereich zuweist. Der Datenbereich kann dem physikalischen IO oder einem anderen logischen Eingang einer anderen Modulinstanz zugewiesen werden.

Das Beispiel ist hier [▶ 37] in der Kurzanleitung Schritt für Schritt beschrieben.

15.3 Beispiel02: Zyklische C++ Logik, die IO vom IO Task verwendet

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, das ein Image eines IO Tasks verwendet.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Quellcode, der vom Assistenten nicht automatisch generiert wird, ist mit einem Beginn-Flag „//sample code“ und Ende-Flag „//sample code end“ gekennzeichnet.
Somit können Sie nach diesen Zeichenketten in den Dateien suchen, um sich ein Bild von den Einzelheiten zu machen.

Beschreibung

Dieses Beispiel beschreibt die Flexibilität von C++ Code bei der Arbeit mit IOs, welche an dem Task konfiguriert sind. Durch diesen Ansatz kann ein kompiliertes C++ Modul flexibler reagieren, wenn mit der IO Task unterschiedlich viele IOs verbunden sind. Eine Einsatzmöglichkeit wäre die zyklische Prüfung analoger Eingangskanäle mit ja nach Projekt unterschiedlicher Anzahl Kanäle.

Das Beispiel enthält

- das C++ Modul „TcIoTaskImageAccessDrv“ mit einer Instanz des Moduls „TcIoTaskIMageAccessDrv_Obj1“
- Ein Task „Task1“ mit einem Image, 10 Eingangsvariablen (Var1..Var10) und 10 Ausgangsvariablen (Var11..Var20).
- Sie sind verbunden: Die Instanz wird von dem Task aufgerufen und die Instanz Nutzt das Image von Task1.

Der C++ Code greift über ein Datenimage auf die Werte zu, das beim Übergang „SAFEOP to OP“ (SO) initialisiert wird.

In der zyklisch ausgeführten Methode „CycleUpdate“ wird durch Aufruf der Helfermethode „CheckValue“ der Wert jeder Eingangsvariable überprüft. Ist er kleiner als 0, wird die entsprechende Ausgangsvariable auf 1 gesetzt, ist er größer als 0, wird sie auf 2 gesetzt und ist sie gleich 0, dann wird der Ausgang auf 3 gesetzt.

Nach der Aktivierung der Konfiguration können Sie auf die Variablen über den Solution Explorer zugreifen und diese setzen.
Doppelklicken Sie auf das Task1-Abbild des Systems, um eine Übersicht zu erhalten.
Die Eingangsvariablen können geöffnet und dann mit dem „Online“ Karteireiter gesetzt werden.

15.4 Beispiel03: C++ als ADS Server

Dieser Artikel beschreibt:

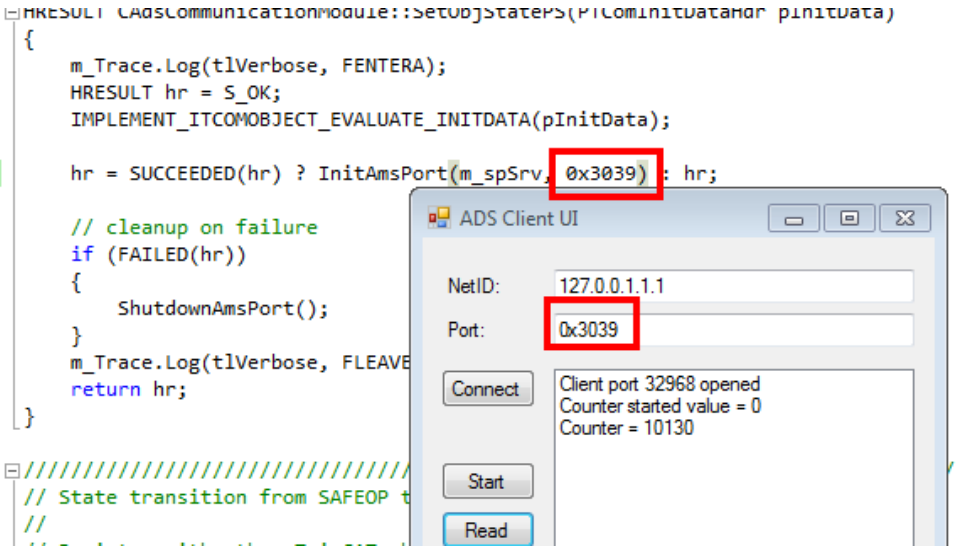
- Erstellung eines TC3 C++ Moduls, das als ADS Server fungiert.
  Der Server stellt eine ADS-Schnittstelle zum Starten / Stoppen / Zurücksetzen einer Zählervariablen im C++ Modul bereit. Der Zähler steht als Modulausgang zur Verfügung und kann einer Ausgangsklemme (analog oder Anzahl digitaler IOs) zugeordnet werden.
  Wie die in C++ geschriebene TC3 ADS Server-Funktionalität zu implementieren ist [▶ 201]

- Einen C# ADS Client erstellen, um mit C++ ADS Server zu interagieren.
  Der Client stellt eine UI zur Verfügung, um lokal oder über Netzwerk mit einem ADS Server mit zu zählender ADS-Schnittstelle verbunden zu werden. Die UI ermöglicht das Starten / Stoppen/ Lesen / Überschreiben und Zurücksetzen des Zählers
  Beispielcode: ADS Client UI geschrieben in C# [▶ 205]

Das Beispiel verstehen

Im Beispiel werden Möglichkeiten zur automatischen Bestimmung eines ADS Ports verwendet. Dies hat den Nachteil, dass der Client bei jedem Start konfiguriert werden muss, um auf den richtigen ADS Port zuzugreifen.

Alternativ kann der ADS Port im Modul, wie unten gezeigt, hartkodiert werden.
Nachteil hier: Das C++ Modul kann nicht mehr als einmal instanziiert werden, da das Teilen eines ADS Ports nicht möglich ist.



15.4.1 Beispiel03: Der in C++ geschriebene TC3 ADS Server

In diesem Artikel wird die Erstellung eines als ADS Server agierenden TC3 C++ Moduls beschrieben.
Der Server stellt eine ADS-Schnittstelle zum Starten / Stoppen / Zurücksetzen einer Zählervariablen im C++ Modul bereit.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Dieses Beispiel beinhaltet ein C++ Modul, das als ADS Server agiert, der einen Zugang zu einem Zähler gewährt, der gestartet, gestoppt und gelesen werden kann.

Die Header-Datei des Moduls definiert die Zählervariable „m_bCount“, und die entsprechende .cpp-Datei initialisiert den Wert im Konstruktor und implementiert die Logik in der „CycleUpdate“ Methode.

Die Methode „AdsReadWriteInd“ in der .cpp-Datei analysiert die eingehenden Meldungen und sendet die Rückgabewerte zurück. Für einen weiteren hinzugefügten Meldungstyp wird ein define in der Header-Datei hinzugefügt.

Details, wie die Definition der ADS-Meldungstypen, werden im folgenden „Kochbuch“ beschrieben, wo Sie das Beispiel manuell zusammenstellen können.

Kochbuch

Nachfolgend werden die einzelnen Schritte der Erstellung des C++ Moduls beschrieben.

1. Erstellen Sie einer neuen TwinCAT 3 Project Solution

Folgen Sie den Schritten für die Erstellung eines neuen TwinCAT 3 Projekts [▶ 37]

2. Erstellen Sie ein C++ Projekt mit ADS Port

Folgen Sie den Schritten für die Erstellung eines neuen TwinCAT 3 C++ Projekts [▶ 39].

Wählen Sie im „Class templates“-Dialog „TwinCAT Module Class with ADS port“.

3. Fügen Sie dem Projekt die Beispiellogik hinzu

6. Öffnen Sie die Header-Datei „<MyClass>.h“ (in diesem Beispiel „Module1.h“) und fügen den Zähler „m_bCount“ als neue Membervariable in den geschützten Bereich hinzu:

```
class CModule1
    : public ITComObject
    , public ITcCyclic
    ,...
{
public:
    DECLARE_IUNKNOWN()
    ....
protected:
    DECLARE_ITCOMOBJECT_SETSTATE();
///<AutoGeneratedContent id="Members">
```

```
    ITcCyclicCallerInfoPtr m_spCyclicCaller;
    .....
///</AutoGeneratedContent>
    ULONG m_ReadByOidAndPid;
    BOOL m_bCount;
};
```

7. Öffnen Sie die Klassendatei „<MyClass>.cpp“ (in diesem Beispiel „Module1.cpp“) und initialisieren die neuen Werte im Konstruktor:

```
CModule1::CModule1()
    .....
{
    memset(&m_Counter, 0, sizeof(m_Counter));
    memset(&m_Inputs, 0, sizeof(m_Inputs));
    memset(&m_Outputs, 0, sizeof(m_Outputs));
    m_bCount = FALSE; // by default the counter should not increment
    m_Counter = 0;    // we also initialize this existing counter
}
```

3.a. Fügen Sie der ADS-Serverschnittstelle die Beispiellogik hinzu.

Normalerweise empfängt der ADS Server eine ADS-Meldung, die zwei Parameter („indexGroup“ und „indexOffset“) und gegebenenfalls weitere Daten „pData“ enthält.

ADS-Schnittstelle entwerfen
Unser Zähler sollte gestartet, gestoppt, zurückgesetzt, mit Wert überschrieben werden oder dem ADS Client auf Anfrage einen Wert liefern:

indexGroup	indexOffset	Beschreibung
0x01	0x01	m_bCount = TRUE, Zähler wird inkrementiert
0x01	0x02	Zählerwert wird ADS Client übergeben
0x02	0x01	m_bCount = FALSE, Zähler wird nicht mehr inkrementiert
0x02	0x02	Zähler zurücksetzen
0x03	0x01	Zähler mit vom ADS Client übergebenen Wert überschreiben

Diese Parameter sind in „modules1Ads.h“ definiert - fügen Sie die blauen Codezeilen ein, um einen neuen Befehl für IG_RESET hinzuzufügen.

```
#include "TcDef.h"
enum Module1IndexGroups : ULONG
{
    Module1IndexGroup1 = 0x00000001,
    Module1IndexGroup2 = 0x00000002, // add command
    IG_OVERWRITE = 0x00000003 // and new command
};

enum Module1IndexOffsets : ULONG
{
    Module1IndexOffset1 = 0x00000001,
    Module1IndexOffset2 = 0x00000002
};
```

Fügen Sie die blauen Codezeilen in Ihre <MyClass>::AdsReadWriteInd() Methode ein (in diesem Falle in Module1.cpp).

```
switch(indexGroup)
{
case Module1IndexGroup1:
    switch(indexOffset)
    {
    case Module1IndexOffset1:
        ...
```

```
        // TODO: add custom code here
        m_bCount = TRUE; // receivedIG=1 IO=1, start counter
        AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 0,NULL);
        break;
    case Module1IndexOffset2:
        ...
        // TODO: add custom code here
        // map counter to data pointer
        pData = &m_Counter;     // received IG=1 IO=2, provide counter value via ADS
     AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 8,pData);
//comment this: AdsReadWriteRes(rAddr, invokeId,ADSERR_NOERR, 0, NULL);
     break;
    }
    break;
case Module1IndexGroup2:
    switch(indexOffset)
    {
    case Module1IndexOffset1:
     ...
     // TODO: add custom code here
    // Stop incrementing counter
    m_bCount = FALSE;
     // map counter to data pointer
    pData = &m_Counter;
     AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 8,pData);
     break;
case Module1IndexOffset2:
    ...
    // TODO: add custom code here
    // Reset counter
    m_Counter = 0;
    // map counter to data pointer
    pData = &m_Counter;
     AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 8, pData);
     break;
}
break;
case IG_OVERWRITE:
    switch(indexOffset)
    {
    case Module1IndexOffset1:
    ...
     // TODO: add custom code here     // override counter with value provided by ADS-client
     unsigned long *pCounter = (unsigned long*) pData;
     m_Counter = *pCounter;
     AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 4, pData);
     break;
}
break;
}
break;
default:
    __super::AdsReadWriteInd(rAddr, invokeId, indexGroup,indexOffset, cbReadLength, cbWriteLength,
pData;
     break;
}
```

3.b. Fügen Sie die Beispiellogik in den zyklischen Teil ein

Die Methode <MyClass>::CycleUpdate() wird zyklisch aufgerufen - das ist die Stelle, wo die Logik zu verändern ist.

```
 // TODO: Replace the sample with your cyclic code
m_Counter+=m_Inputs.Value; // replace this line
m_Outputs.Value=m_Counter;
```

In diesem Falle wird der Zähler **mCounter** inkrementiert, wenn die boolsche Variable **m_bCount** TRUE ist.

Fügen Sie diesen If-Fall in Ihre zyklische Methode ein.

```
HRESULT CModule1::CycleUpdate(ITcTask* ipTask,
ITcUnknown* ipCaller, ULONG context)
{
HRESULT hr = S_OK;
// handle pending ADS indications and confirmations
CheckOrders();
....
```

```
 // TODO: Replace the sample with your cyclic code
if (m_bCount)    // new part
{
    m_Counter++;
}
m_Outputs.Value=m_Counter;
}
```

4. Server-Beispiel ausführen

8. Den TwinCAT TMC Code Generator [▶ 44] ausführen, um die Ein-/Ausgänge für das Modul bereitzustellen.
9. Diese ist möglicherweise ein guter Zeitpunkt, um das Projekt zu speichern...
10. Das Projekt kompilieren [▶ 44]
11. Eine Modulinstanz erstellen [▶ 46]
12. Einen zyklischen Task erstellen [▶ 48] und das C++ Modul für die Ausführung in diesem Kontext konfigurieren.
13. Sie können optional die Hardware IO scannen und das Symbol „Value“ von Ausgängen bestimmten Ausgangsklemmen zuordnen.
14. Das TwinCAT Projekt aktivieren [▶ 52]

5. Finden Sie den ADS Port der Modulinstanz heraus

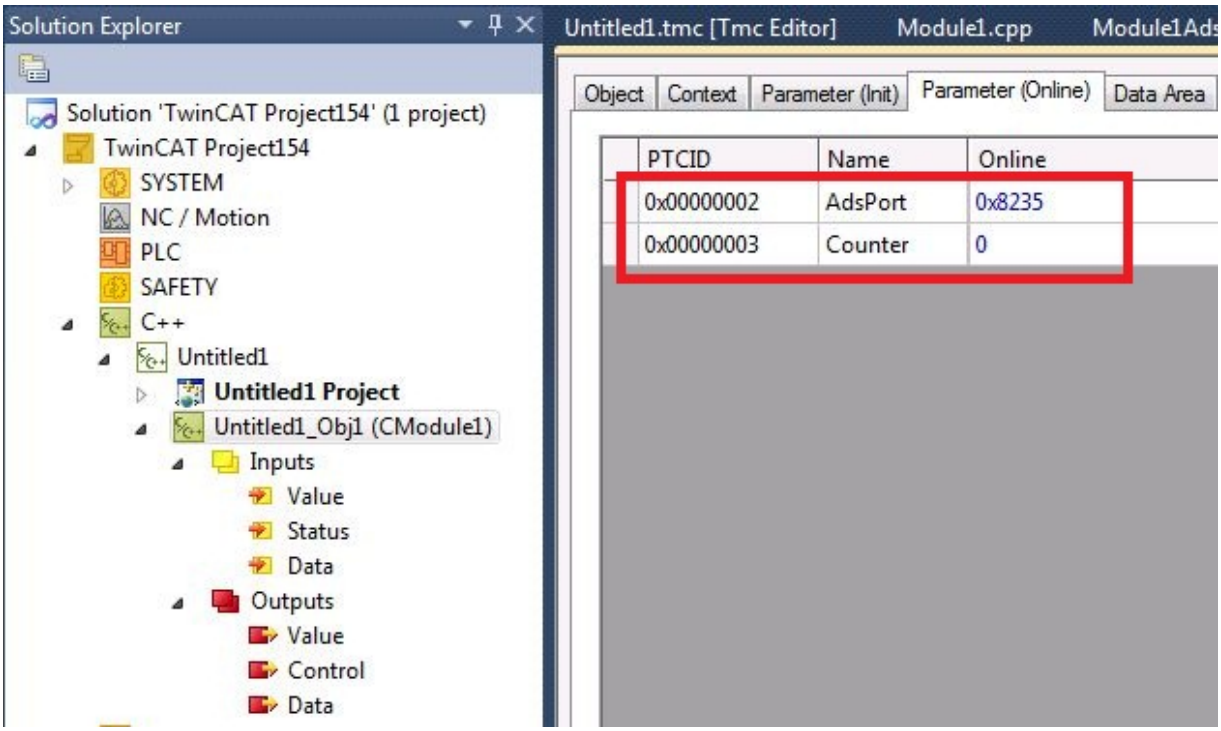
Im Allgemeinen kann der ADS Port

- vornummeriert sein, damit es immer der gleiche Port für diese Modulinstanz ist
- flexibel gehalten werden, um mehreren Instanzen des Moduls die Möglichkeit zu bieten, dass ihnen beim Start des TwinCAT Systems ihr eigener ADS Port zugewiesen wird.

In diesem Beispiel haben wir uns für die Standardeinstellung (flexibel halten) entschieden - also müssen wir den ADS Port herausfinden, der dem Modul, das wird soeben aktiviert haben, zugewiesen wurde:

15. Zu Modulinstanz navigieren, Karteireiter „Parameter Online“ auswählen.
    ADSPort
    Es ist 0x8235 oder Dezimal 33333 zugewiesen (lustig für ein Beispiel, kann aber bei Ihrem Beispiel anders sein). Werden mehr und mehr Instanzen erstellt, erhält jede Instanz ihren eigenen eindeutigen AdsPort.

Zähler
Immer noch auf „0“, da wir die ADS-Meldung zum Starten der Inkrementierung nicht gesendet haben.



⇨ Der Server-Teil ist abgeschlossen - fahren wir also fort mit ADS Client sendet die ADS-Meldungen [▶ 205].

15.4.2 Beispiel03: ADS Client UI in C#

Dieser Artikel beschreibt den ADS Client, der ADS-Meldungen an den zuvor beschriebenen ADS Server sendet.

Die Implementierung des ADS Servers hängt weder von der Sprache (C++ / C# / PLC / ...), noch von der TwinCAT Version (TwinCAT 2 oder TwinCAT 3) ab.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

✓ Dieser Code erfordert .NET Framework 3.5 oder höher!

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das Projekt rechtsklicken und „Build“ klicken)
4. Programm starten, d.h. Rechtsklick auf Projekt, Debug->Start new instance

Beschreibung

Dieser Client führt zwei Aufgaben aus:

- Den zuvor beschriebenen ADS Server testen.
- Beispielcode für die Implementierung eines ADS Client bereitstellen.

Den Client benutzen

Kommunikationspartner auswählen

Die beiden ADS-Parameter eingeben, um Ihren ADS-Kommunikationspartner zu bestimmen

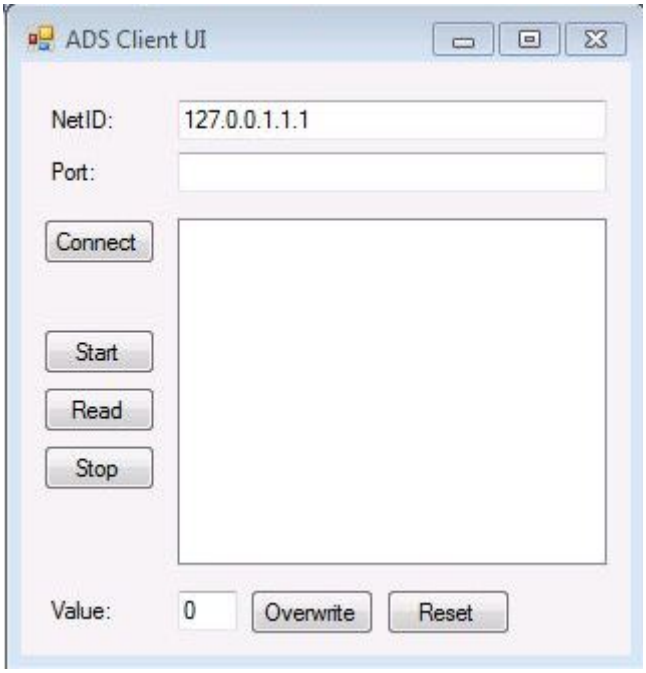
- NetID:
  127.0.0.1.1.1 (für ADS-Partner auch mit lokalem ADS Message Router verbunden)

  Geben Sie eine andere NetID ein, wenn Sie über das Netzwerk mit einem an einen anderen ADS Router angeschlossenen ADS-Partner kommunizieren möchten.
  Zuvor müssen Sie einmal eine ADS Route zwischen Ihrem Gerät und dem fernen Gerät herstellen.

- AdsPort
  Geben Sie den AdsServerPort Ihres Kommunikationspartners ein
  Verwechseln Sie nicht den ADS Server Port (der ausdrücklich Ihren eigenen Message-Handler implementiert hat) mit dem regulären ADS Port zwecks Zugriff auf Symbole (es gibt nichts zu tun, wird automatisch zur Verfügung gestellt).
  Finden Sie den zugewiesenen AdsPort [▶ 201] in diesem Beispiel heraus, der AdsPort war 0x8235 (dez 33333).

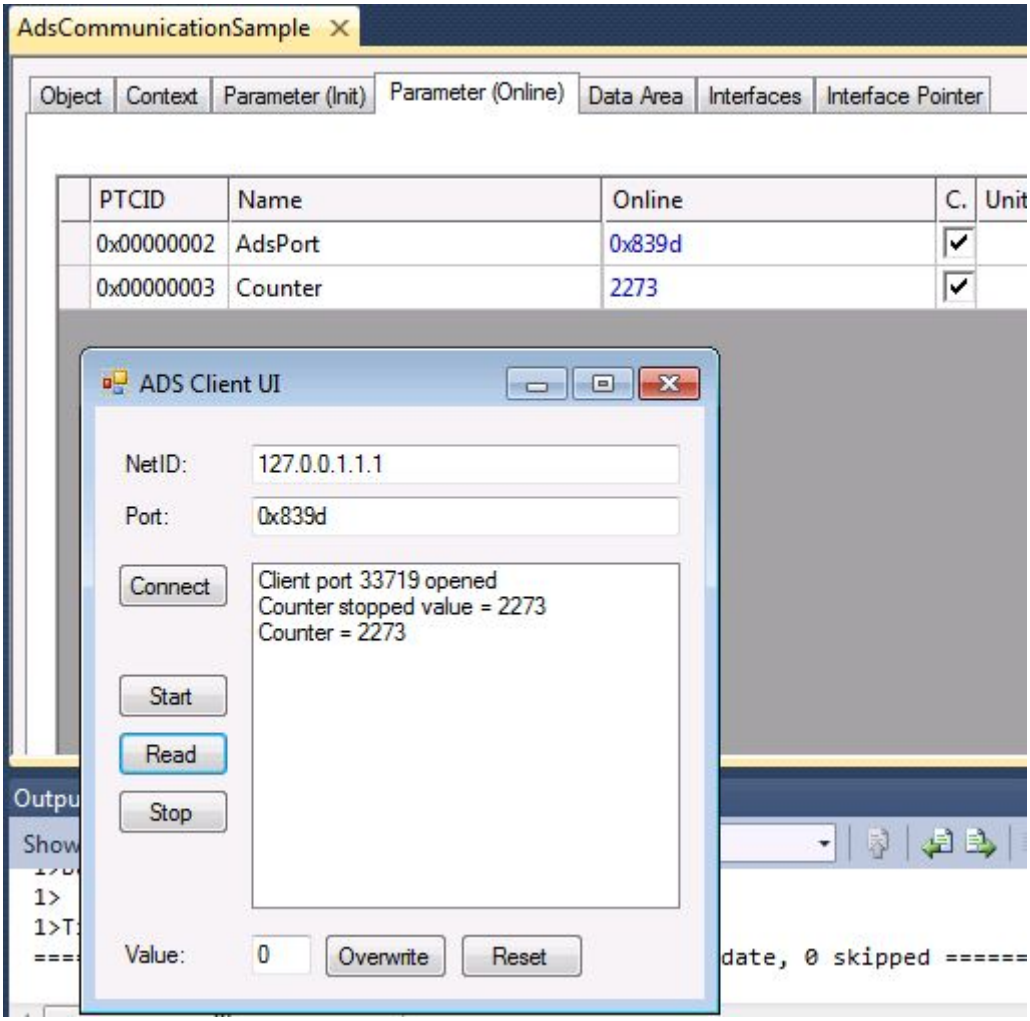
Verbindung mit Kommunikationspartner herstellen

Beim Klicken auf „Connect“ wird die Methode TcAdsClient.Connect zwecks Herstellung einer Verbindung mit dem konfigurierten Port aufgerufen.



Mit Hilfe der Schaltflächen Start / Lesen / Stopp / Überschreiben / Zurücksetzen werden ADS-Meldungen an den ADS Server gesendet.
Die spezifischen indexGroup / indexOffset Befehle wurden bereits in der ADS-Schnittstelle des ADS Servers entworfen [▶ 201].

Das Ergebnis des Klickens auf die Befehlsschaltflächen ist auch in der Modulinstanz auf der Registerkarte „Parameter (Online)“ zu sehen.



C# Programm

Hier ist der „Kern“ Code des ADS Client - für die GUI usw., bitte ZIP-Datei oben herunterladen.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using TwinCAT.Ads;

namespace adsClientVisu
{
public partial class form : Form
{
    public form()
    {
     InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
     // create a new TcClient instance
     _tcClient = new TcAdsClient();
     adsReadStream = new AdsStream(4);
     adsWriteStream = new AdsStream(4);
    }

    /*
     * Connect the client to the local AMS router
```

```
 */

private void btConnect_Click(object sender, EventArgs e)
{
 AmsAddress serverAddress = null;
 try
 {
     serverAddress = new AmsAddress(tbNetId.Text,
                     Int32.Parse(tbPort.Text));
 }
 catch
 {
     MessageBox.Show("Invalid AMS NetId or Ams port");
     return;
 }

 try
 {
     _tcClient.Connect(serverAddress.NetId, serverAddress.Port);
     lbOutput.Items.Add("Client port " + _tcClient.ClientPort + " opened");

 }
 catch
 {
     MessageBox.Show("Could not connect client");
 }
}

private void btStart_Click(object sender, EventArgs e)
{
 try
 {
     _tcClient.ReadWrite(0x1, 0x1, adsReadStream, adsWriteStream);
     byte[] dataBuffer = adsReadStream.ToArray();
     lbOutput.Items.Add("Counter started value = " + BitConverter.ToInt32(dataBuffer, 0));
 }

 catch (Exception err)
 {
     MessageBox.Show(err.Message);
 }

}

private void btRead_Click(object sender, EventArgs e)
{
 try
 {
     _tcClient.ReadWrite(0x1, 0x2, adsReadStream, adsWriteStream);
     byte[] dataBuffer = adsReadStream.ToArray();
     lbOutput.Items.Add("Counter = " + BitConverter.ToInt32(dataBuffer, 0));
 }

 catch (Exception err)
 {
     MessageBox.Show(err.Message);
 }
}

private void btStop_Click(object sender, EventArgs e)
{
 try
 {
     _tcClient.ReadWrite(0x2, 0x1, adsReadStream, adsWriteStream);
     byte[] dataBuffer = adsReadStream.ToArray();
     lbOutput.Items.Add("Counter stopped value = " + BitConverter.ToInt32(dataBuffer, 0));
 }

 catch (Exception err)
 {
     MessageBox.Show(err.Message);
 }
}

private void btReset_Click(object sender, EventArgs e)
{
 try
 {
     _tcClient.ReadWrite(0x2, 0x2, adsReadStream, adsWriteStream);
     byte[] dataBuffer = adsReadStream.ToArray();
```

```
            lbOutput.Items.Add("Counter reset Value = " + BitConverter.ToInt32(dataBuffer, 0));
        }

        catch (Exception err)
        {
            MessageBox.Show(err.Message);
        }
    }
}
}
```

15.5 Beispiel05: C++ CoE Zugriff über ADS

In diesem Artikel wird die Implementierung eines TC3 C++ Moduls beschrieben, das auf das CoE (CANopen over EtherCAT) Register einer EtherCAT Klemme zugreifen kann.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Beispiel wie unten beschrieben konfigurieren
6. Die Konfiguration aktivieren

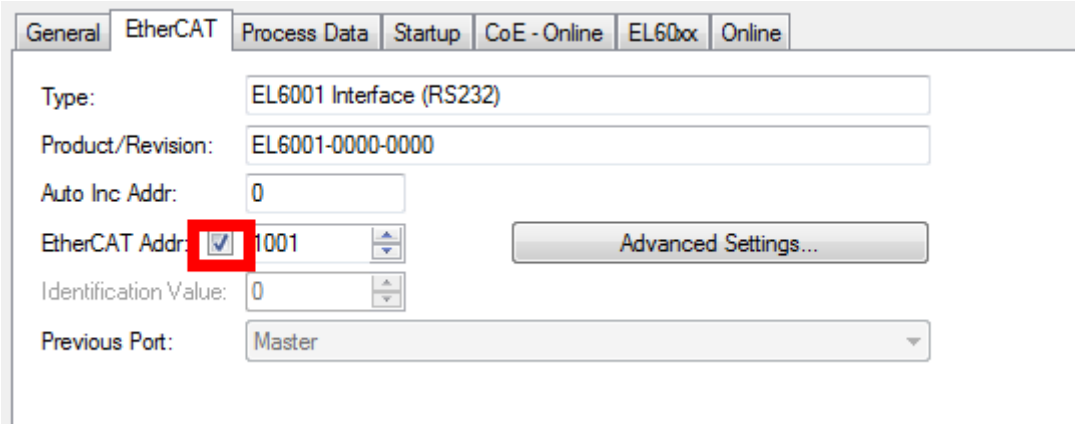
Beschreibung

Dieses Beispiel beschreibt den Zugriff auf eine EtherCAT-Klemme, der die Hersteller-ID liest und die Baudrate für die serielle Kommunikation festlegt.

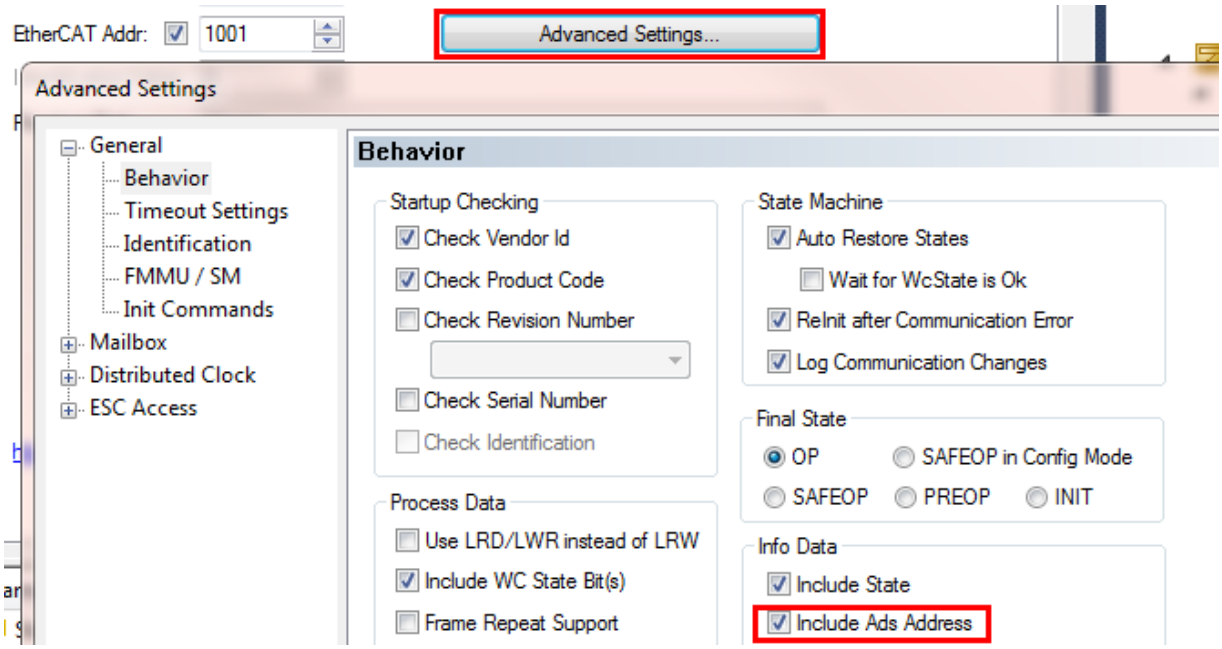
Dieses Beispiel beschreibt den Schnellstart zwecks Erstellung eines C++ Moduls, das einen Zähler bei jedem Zyklus inkrementiert und den Zähler dem logischen Ausgang „Value“ im Datenbereich zuweist.

Konfiguration

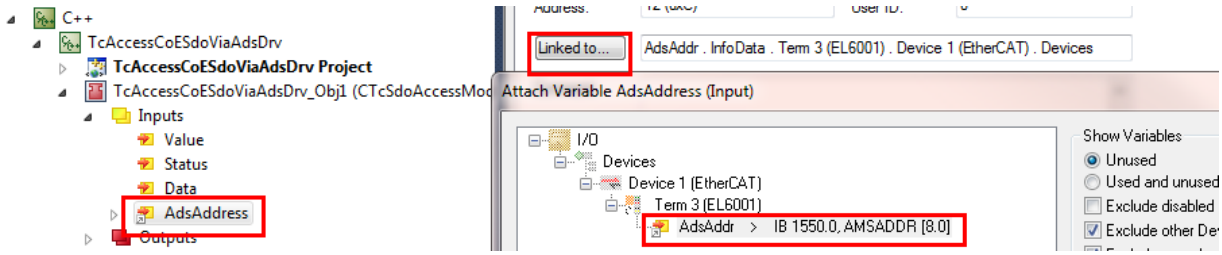
7. Die EtherCAT-Adresse der betreffenden Klemme aktivieren und zuweisen.



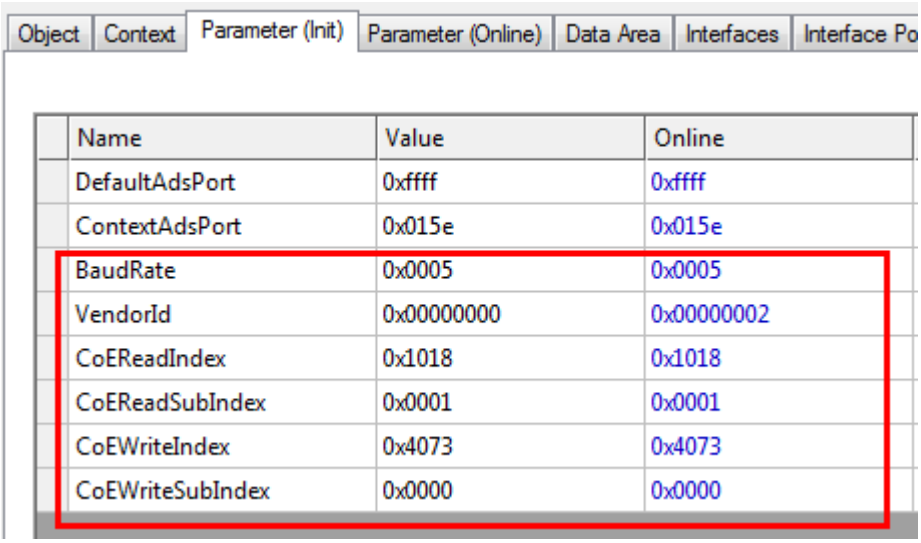
8. Den Einschluss der ADS-Adresse in den erweiterten Einstellungen der EtherCAT-Klemme aktivieren:



9. Die ADS-Adresse (einschließlich netId und port) dem Moduleingang AdsAdress zuweisen:



10. Der Modulparameter wird im Verlauf der Initialisierung festgelegt:



Name	Value	Online
DefaultAdsPort	0xffff	0xffff
ContextAdsPort	0x015e	0x015e
BaudRate	0x0005	0x0005
VendorId	0x00000000	0x00000002
CoEReadIndex	0x1018	0x1018
CoEReadSubIndex	0x0001	0x0001
CoEWriteIndex	0x4073	0x4073
CoEWriteSubIndex	0x0000	0x0000

15.6 Beispiel06: UI-C#-ADS Client lädt die Symbolik vom Modul hoch

Dieser Artikel beschreibt die Implementierung eines ADS Client um

- mit einem ADS Server, der ein Prozessabbild (Datenbereich) bereitstellt, in Verbindung zu treten
  Die Verbindung kann lokal oder fern über Netzwerk hergestellt werden.

- Symbolinformation hochzuladen
- Daten synchron zu lesen / schreiben
- Symbole abonnieren, um Werte „bei Veränderung“ als Callback zu erhalten.

Source

Erhalten Sie den Quellcode für dieses Client-Beispiel:

✓ Dieser Code erfordert .NET Framework 3.5 oder höher!

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das Projekt rechtsklicken und „Build“ klicken)
4. Programm starten, d.h. Rechtsklick auf Projekt, Debug->Start new instance

Das Client-Beispiel sollte mit Beispiel 03 „TC3 C++ ADS Server functionalities“ verwendet werden.

Bitte öffnen Sie Beispiel 03 bevor Sie mit diesem client-seitigen Beispiel beginnen!

Beschreibung

Anhand von diesem Beispiel werden die Möglichkeiten von ADS beschrieben. Es zeigt wie

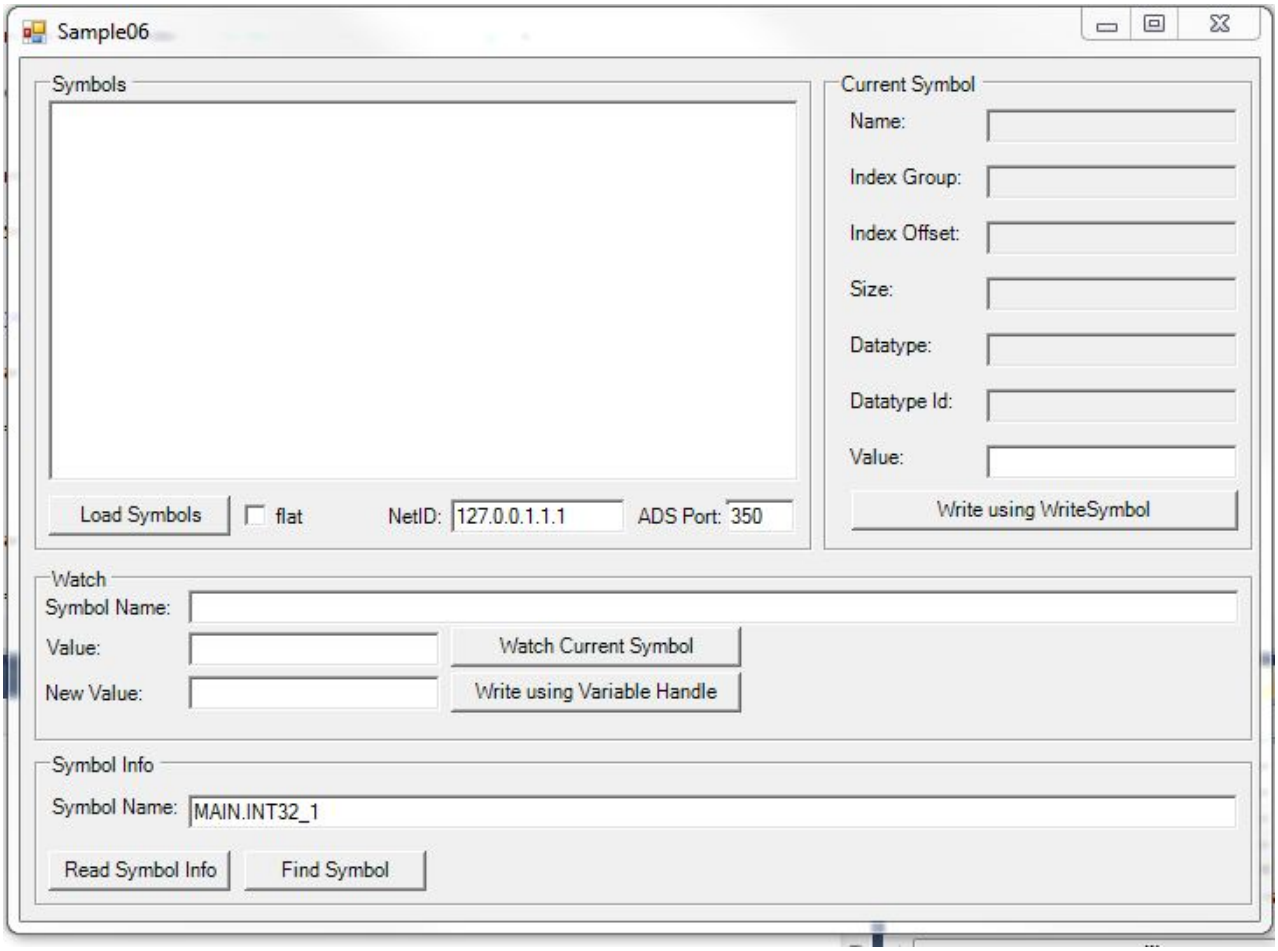
Die Einzelheiten der Implementierung werden in „Form1.cs“, das im Download enthalten ist, beschrieben. Die Verbindung über ADS mit dem Zielsystem wird in der „btnLoad_Click“ Methode, die beim Klicken auf die „Load Symbols“ Schaltfläche aufgerufen wird, hergestellt. Von dort aus können Sie die verschiedenen Funktionen der GUI erkunden.

Hintergrundinformation:

Diesem ADS Client ist es Einerlei, ob der ADS Server auf TwinCAT2 oder auf TwinCAT 3 basiert. Auch spielt es keine Rolle, ob der Server ein C++ Modul, SPS-Modul oder IO Task ohne jede Logik ist.

Die ADS Client UI

Nach dem Starten des Beispiels wird die Benutzerschnittstelle (UI) eingeblendet.



Kommunikationspartner auswählen

Nach dem Starten des Client, die beiden ADS-Parameter eingeben, um Ihren ADS-Kommunikationspartner zu bestimmen

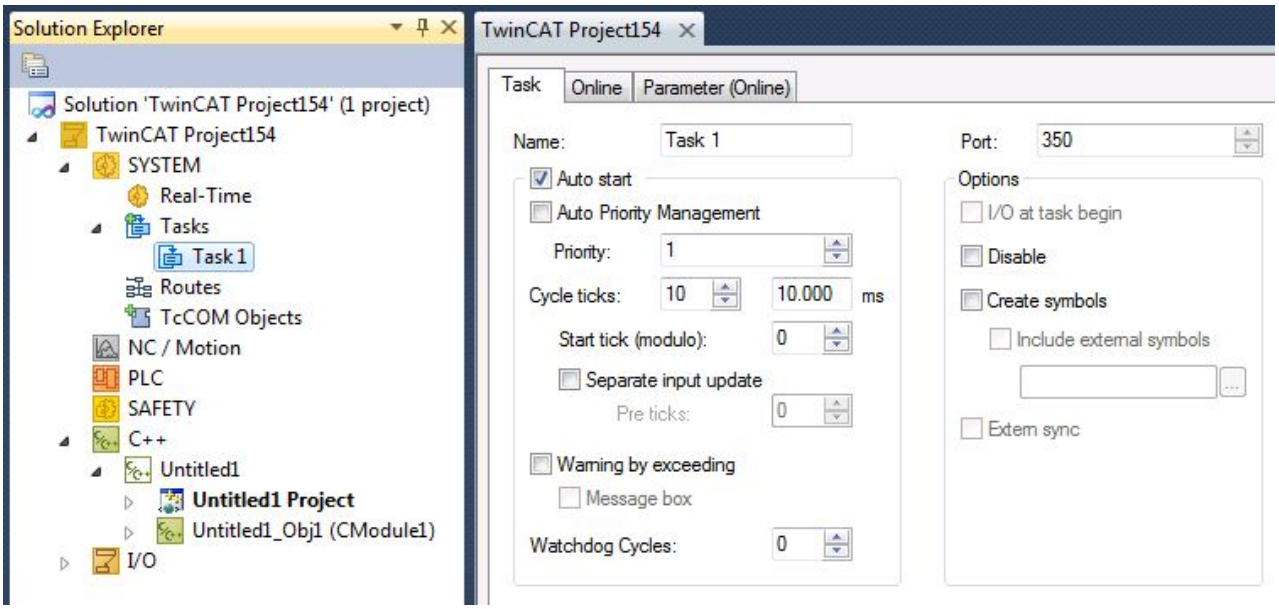
- NetID:
  127.0.0.1.1.1 (für ADS-Partner auch mit lokalem ADS Message Router verbunden)

  Geben Sie eine andere NetID ein, wenn Sie über das Netzwerk mit einem an einen anderen ADS Router angeschlossenen ADS-Partner kommunizieren möchten.
  Zuvor müssen Sie einmal eine ADS Route [▶ 180] zwischen Ihrem Gerät und dem fernen Gerät herstellen.

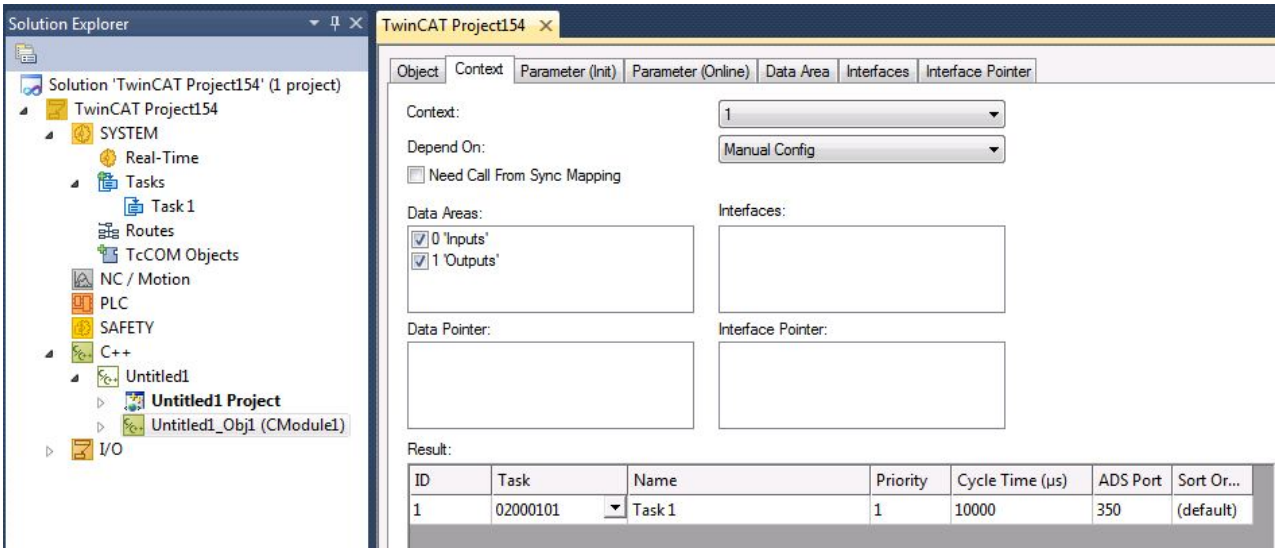
- AdsPort
  Geben Sie den AdsPort Ihres Kommunikationspartners ein: 350 (in diesem Beispiel)

Hinweis	**Nicht den ADS Server Port mit dem regulären ADS Port verwechseln.** Verwechseln Sie nicht den ADS Server Port (der ausdrücklich in Beispiel 03 implementiert wurde, um Ihren eigenen Message-Handler bereitzustellen) mit dem regulären ADS Port zwecks Zugriff auf Symbole (es gibt nichts zu tun, wird automatisch zur Verfügung gestellt): Wir benötigen den regulären ADS Port, um auf Symbole zugreifen zu können. Sie können den AdsPort beim IO Task Ihrer Instanz oder bei der Modulinstanz selber herausfinden (weil das Modul im Kontext des IO Tasks ausgeführt wird).

Navigieren Sie zum IO Task „Task1“ und achten auf den Wert von Port: 350



Weil die C++ Modulinstanz im Kontext von Task1 ausgeführt wird, ist der ADS Port auch gleich 350.

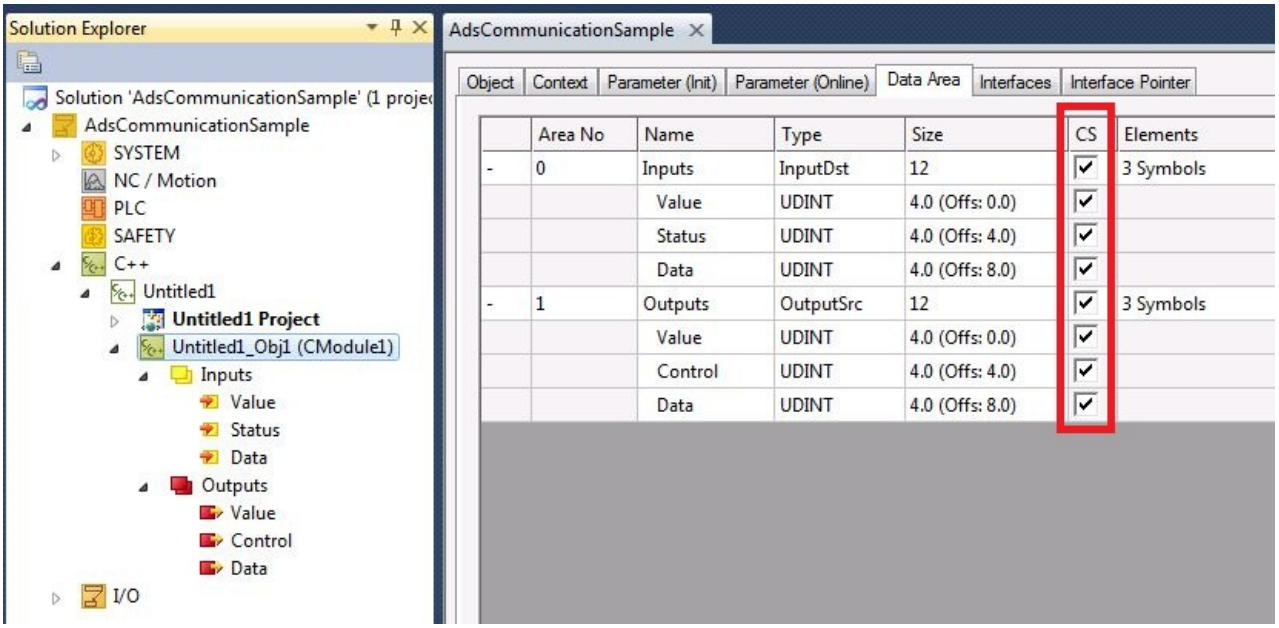


Aktivierte Symbole für Zugriff über ADS verfügbar

Es besteht die Möglichkeit einzelne Symbole oder gar vollständige Datenbereiche für den Zugriff über ADS bereitzustellen oder eben nicht bereitzustellen.
Navigieren Sie zur der Registerkarte „Data Area“ Ihrer Instanz und aktivieren/deaktivieren die Spalte „C/S“.

In diesem Beispiel sind alle Symbole gekennzeichnet und demzufolge für den ADS-Zugriff verfügbar.

Nach den Änderungen bitte die „Konfiguration aktivieren“.



Symbole laden

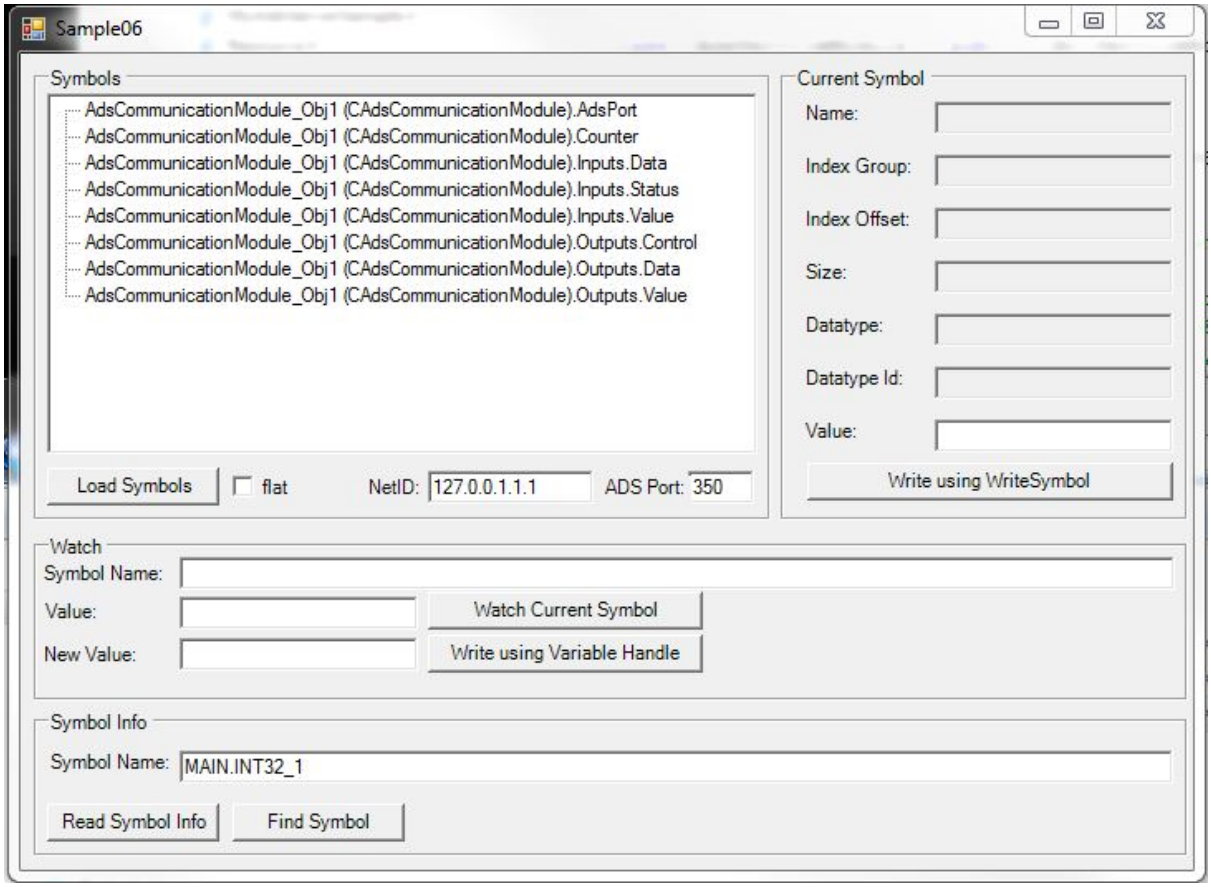
Nach der Einrichtung der NetID und des ADS Port auf die Schaltfläche „Load Symbols“ klicken, um eine Verbindung mit dem Zielsystem herzustellen und die Symbole zu laden.

Daraufhin sind alle verfügbaren Symbole zu sehen. Anschließend können Sie:

- Einen neuen Wert schreiben:
  Ein Symbol im linken Baum auswählen, z.B. „Counter“
  Einen neuen Wert in das Bearbeitungsfeld „Value“ auf der rechten Seite eingeben und auf „Write using WriteSymbol“ klicken.
  Der neue Wert wird in den ADS Server geschrieben.

  Nach dem Schreiben eines neuen Werts mit „Write using WriteSymbol“ erhält auch die C# Anwendung einen Callback mit dem neuen Wert.

- Abonnieren, um einen Callback bei Werteveränderungen zu erhalten.
  Ein Symbol im linken Baum auswählen, z.B. „Counter“
  Auf „Watch Current Symbol“ klicken



15.7 Beispiel07: Empfang von ADS Notifications

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, das ADS Notifications bezüglich Datenänderungen auf anderen Modulen empfängt.
Weil jede andere ADS-Kommunikation auf ähnliche Weise implementiert werden muss, kann dieses Beispiel als allgemeiner Einstiegspunkt für die Initialisierung der ADS-Kommunikation von TwinCAT C++ Modulen aus betrachtet werden.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Dieses Beispiel beschreibt den Empfang von ADS Notifications in einem TwinCAT C++ Modul.

Hierfür enthält die Solution 2 Module.

- Ein C++ Modul, das sich für die Abfrage von ADS Notifications einer Variablen anmeldet.

- Zum einfachen Verständnis: Ein SPS-Programm, das eine Variable „MAIN.PlcVar“ bereitstellt. Wenn ihr Wert sich verändert, wird eine ADS Notification an das C++ Modul geschickt.
- Das C++ Modul nutzt die Möglichkeiten Meldungen aufzuzeichnen - also zum Verständnis des Codes, starten Sie einfach das Beispiel und achten dann auf den Ausgang / Fehler-Log, wenn Sie den Wert „Main.PlcVar“ des SPS-Moduls ändern.

Die Adresse wird beim Modulübergang PREOP->SAFEOP („SetObjStatePS“) vorbereitet. Die „CycleUpdate“ Methode beinhaltet eine einfache Zustandsmaschine, die den erforderlichen ADS-Befehl sendet. Entsprechende Methoden zeigen die Empfangsbestätigungen an.

Die geerbte und überladene Methode „AdsDeviceNotificationInd“ wird beim Eingang einer Notification aufgerufen.

Während der Abschaltung werden ADS-Meldungen beim Übergang zwecks Abmeldung versendet („SetObjStateOS“) und das Modul wartet bis zum Auftreten einer Zeitüberschreitung auf den Eingang von Bestätigungen.

Hinweis	**Beginn der Modulentwicklung** Erstellen eines TwinCAT C++ Moduls mit Hilfe des ADS Port-Assistenten. Dadurch wird alles, was Sie zum Aufbau einer ADS-Kommunikation benötigen, eingerichtet. Einfach die notwendigen ADS-Methoden von „ADS.h“, wie im Beispiel gezeigt, verwenden und überschreiben.

Siehe auch

ADS-Kommunikation [▶ 155]

15.8 Beispiel10: Modulkommunikation: Verwendung von Datenzeigern

Dieser Artikel beschriebt die Implementierung von zwei TC3 C++ Modulen, die über einen Datenzeiger kommunizieren.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Diese Kommunikation baut auf einen „geteilten“ Datenbereich auf: Von einem Modul bereitgestellt und über Zeiger von einem anderen Modul aus erreichbar.

Es ist nicht möglich, dass zwei verschiedene Datenzeiger mit dem selben Eintrag in einen Ausgangs- oder Eingangsdatenbereich verknüpft sind, weil es ohne diese Limitierung zu Synchronisationsproblemen käme. Aus diesem Grunde sammelt ein ModuleDataProvider Modul Ein- und Ausgang in einen Standarddatenbereich, der nicht dieser Einschränkung unterliegt.

Alles in allem beinhaltet dieses Beispiel die folgenden Module:

- ModuleDataProvider stellt einen Datenbereich zur Verfügung, auf den andere Module zugreifen können.
  Der Datenbereich enthält 4 Bits (2 werden für Eingang, 2 für Ausgang verwendet) und 2 Integer (einen für Eingang, einen für Ausgang).

- ModuleDataInOut stellt „normale“ Eingangsvariablen, die in den Datenbereich des ModuleDataProvider geschrieben werden, und auch Ausgangsvariablen, die aus dem Datenbereich gelesen werden, zur Verfügung.
  Diese Instanz der Klasse CModuleDataInOut fungiert als eine Simulation für realen IO.
- ModuleDataAccessA greift auf den Datenbereich vom ModuleDataProvider zu und bearbeitet Bit1 / BitOut1 und die Integer auf zyklische Weise.
- ModuleDataAccessB greift auf den Datenbereich vom ModuleDataProvider zu und bearbeitet Bit2 / BitOut2 und die Integer auf zyklische Weise.

Der Nutzer des Beispiels triggert ModuleDataInOut mittels Setzen der Variablen ValueIn / Bit1 / Bit2:

- Beim Setzen des Eingangs „Bit1“ wird der Ausgang „Switch1“ entsprechend gesetzt.
- Beim Setzen des Eingangs „Bit2“ wird der Ausgang „Switch2“ entsprechend gesetzt.
- Beim Setzen des Eingangs „ValueIn“ wird der Ausgang „ValueOut“ bei jedem Zyklus zweimal inkrementiert.

Alle Module sind so konfiguriert, dass sie den gleichen Taskkontext haben, was nötig ist, weil der Zugriff über Zeiger keinerlei Synchronisierungsmechanismus bietet. Die Reihenfolge der Ausführung entspricht derjenigen, die auf der Registerkarte Kontextkonfiguration festgelegt wurde. Dieser Wert wird als Parameter „SortOrder“ weitergegeben und im Smart Pointer des zyklischen Aufrufers (m_spCyclicCaller), in dem auch die Objekt-ID des zyklischen Aufrufers enthalten ist, gespeichert.

Das Modul ModuleDataInOut hat Ein- und Ausgangsvariablen. Diese sind mit den entsprechenden Variablen des Datenanbieters verknüpft.

Das Modul ModuleDataProvider stellt einen Eingangs- und einen Ausgangsdatenbereich zur Verfügung und implementiert die Schnittstelle ITcIoCylic. Die Methode InputUpdate kopiert Daten von den Eingangsvariablen auf das DataIn Symbol des standardmäßigen Datenbereichs „Data“ und die Methode OutputUpdate kopiert Daten vom DataOut Symbol auf die Ausgangsvariablen.

Die Module ModuleDataAccessA und ModuleDataAccessB besitzen Zeiger auf Datenbereiche des Datenanbieters über Verknüpfungen. Diese Zeiger werden beim Übergang SAFEOP zu OP initialisiert. ModuleDataAccessA setzt das BitOut1 entsprechend Bit1 auf zyklische Weise. ModuleDataAccessB entsprechend mit BitOut2 / Bit2. Beide inkrementieren ValueOut mittels Multiplikation des internen Zählers mit dem Wert ValueIn.

15.9 Beispiel11: Modulkommunikation: SPS-Modul ruft eine Methode eines C-Moduls auf

Dieser Artikel beschreibt die Implementierung:

- Eines C++ Moduls, das Methoden für die Steuerung einer Zustandsmaschine zur Verfügung stellt. Folgen Sie dieser schrittweisen Einführung bezüglich der Implementierung eines C++ Moduls, das eine Schnittstelle zur Zustandsmaschine zur Verfügung stellt.

- Ein SPS-Modul um die Funktionalität des C++ Moduls aufzurufen
  Dass keine hartkodierte Verknüpfung zwischen der SPS und dem C++ Modul existiert, ist ein großer Vorteil. Stattdessen kann die aufgerufene C++ Instanz im Systemmanager konfiguriert werden. Folgen Sie dieser schrittweisen Einführung bezüglich der Implementierung eines SPS Projekts, das Methoden von einem C++ Modul aufruft.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf beide Projekte rechtsklicken und auf „Build“ klicken)
4. Das Programm starten - auf die „activate configuration“ Schaltfläche in TwinCAT 3 drücken

15.9.1 Beispiel11: Eine Methode zur Verfügung stellendes TwinCAT 3 C++ Modul

Dieser Artikel beschreibt die Erstellung eines TwinCAT 3 C++ Moduls, das eine Schnittstelle mit einigen Methoden zur Verfügung stellt, die von einer SPS, aber auch von anderen C++ Modulen aufgerufen werden kann.

Die Idee besteht darin, eine einfache Zustandsmaschine in C++ zu erstellen, die von anderen Modulen von außen gestartet und gestoppt werden kann, aber auch das Setzen oder Lesen des bestimmten Zustands der C++ Zustandsmaschine ermöglicht.

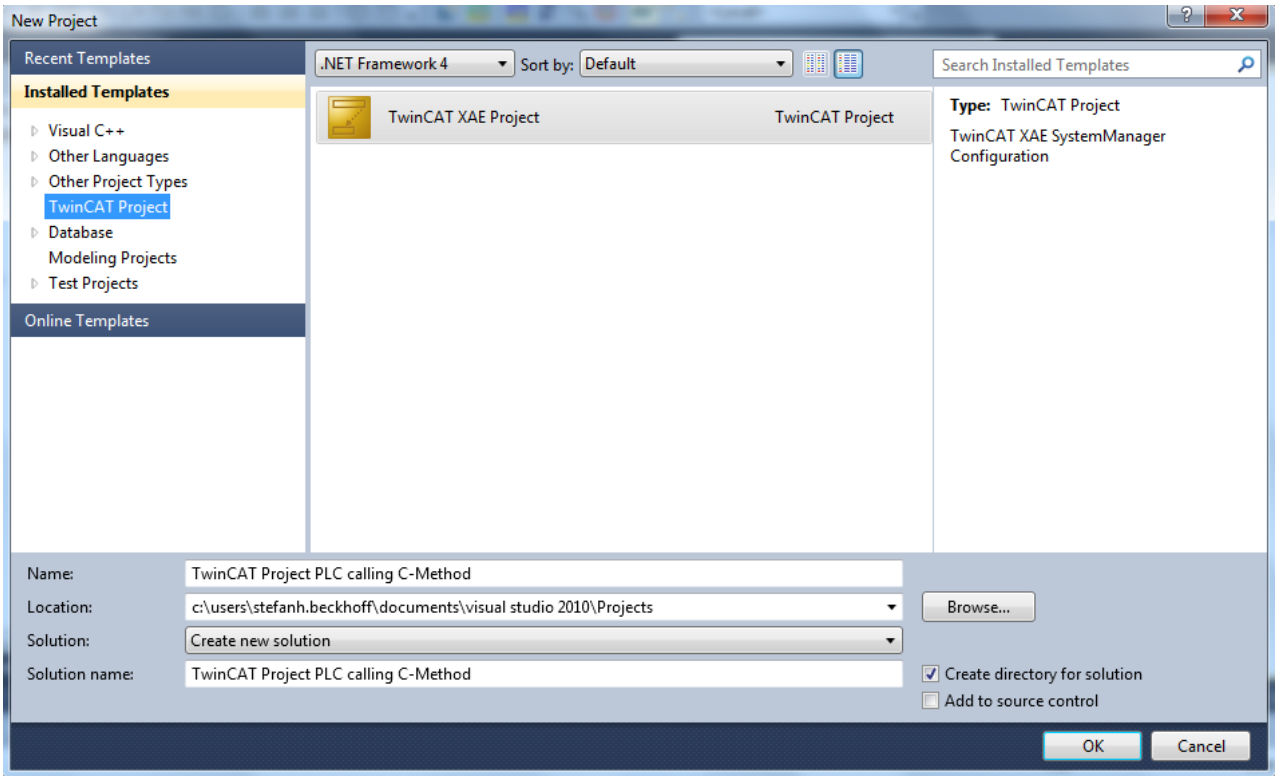
Zwei weitere Artikel nutzen das Ergebnis dieser C++ Zustandsmaschine.

- Aufruf der Funktionalität von der SPS Logik her [▶ 217] - also von der SPS auf C++ Code einwirken
- Aufruf der Funktionalität von der C++ Logik [▶ 247] - also Wechselwirkung zwischen zwei C++ Modulen

Dieser Artikel beschreibt:

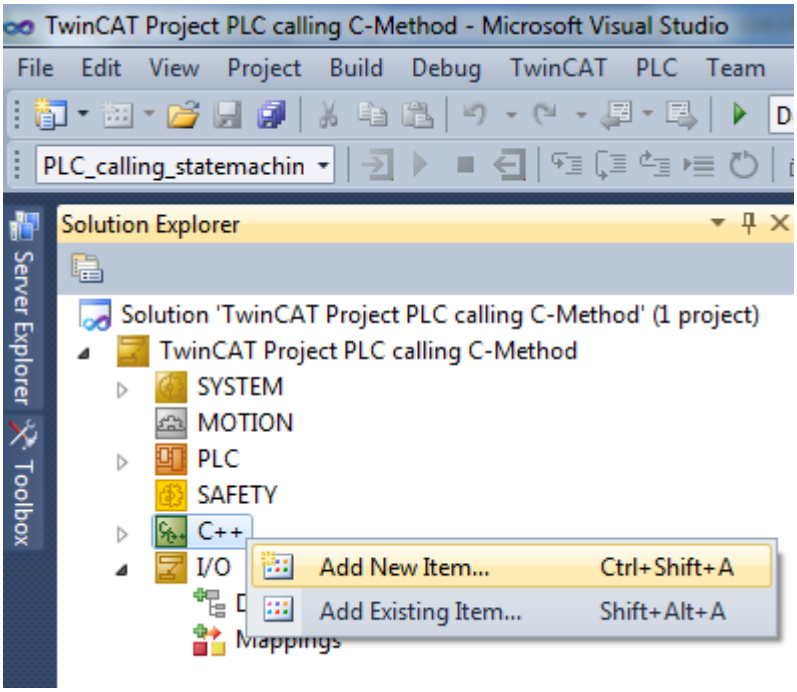
- Schritt 1: Erstellen Sie ein neues TwinCAT 3 Projekt [▶ 219]
- Schritt 2: Erstellen Sie einen neuen TwinCAT 3 C++ Treiber [▶ 219]
- Schritt 3: Eine neue TwinCAT 3 Schnittstelle erzeugen [▶ 221]
- Schritt 4: Fügen Sie der Schnittstelle Methoden hinzu [▶ 222]
- Schritt 5: Neue Schnittstelle zum Modul hinzufügen [▶ 225]
- Schritt 6: Starten Sie den TwinCAT TMC Code Generator um einen Code für die Modulklassenbeschreibung zu erzeugen [▶ 227]
- Schritt 7: Implementierung der Membervariablen und des Konstruktors [▶ 227]
- Schritt 8: Methoden implementieren [▶ 228]
- Schritt 9: Zyklische Aktualisierung implementieren [▶ 228]
- Schritt 10: Code kompilieren [▶ 229]
- Schritt 11: Eine Instanz des C++ Moduls erstellen [▶ 230]
- Schritt 12: Fertig, Ergebnisse überprüfen [▶ 231]

Schritt 1: TwinCAT 3 Projekt erstellen

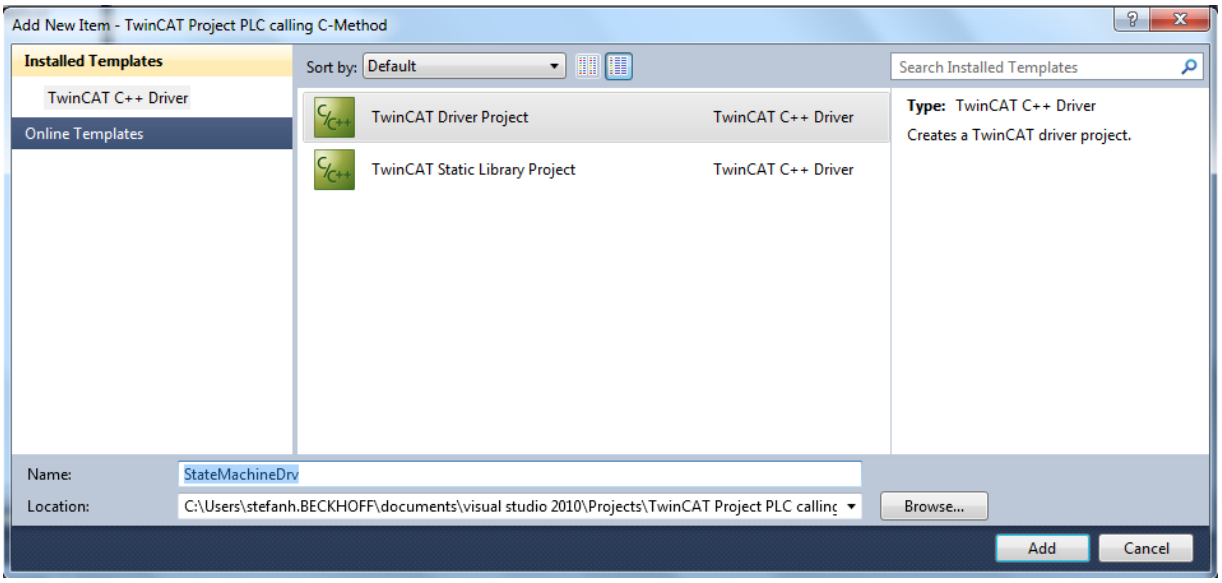


Schritt 2: Erstellen Sie einen neuen TwinCAT 3 C++ Treiber

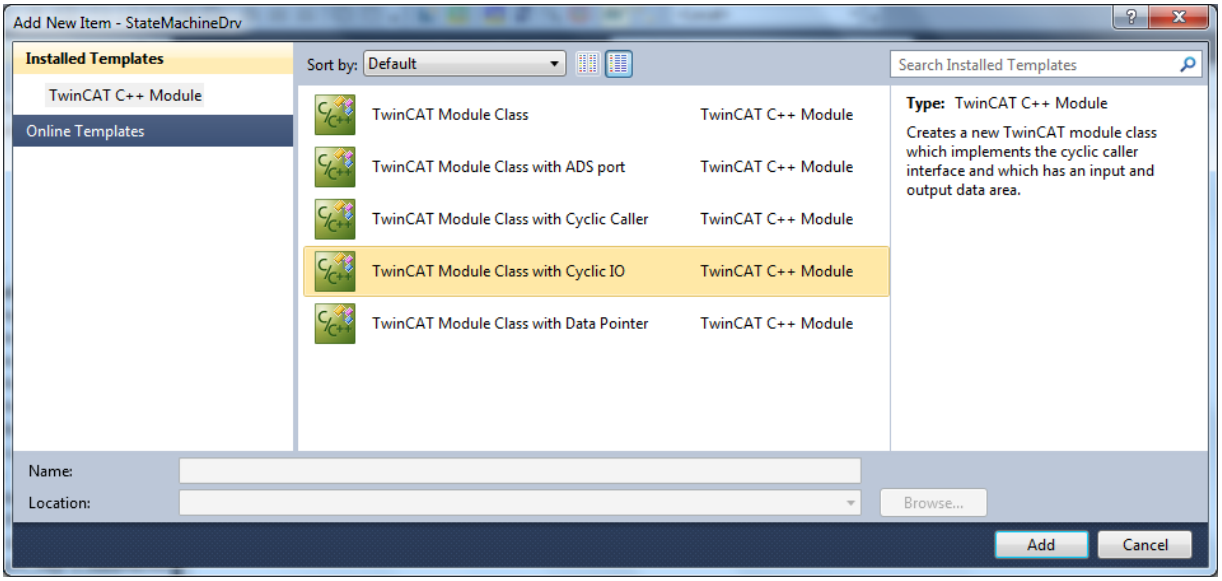
1. Klicken Sie mit der rechten Maustaste auf „C++“ und „Add New Item...“ auswählen.



2. Wählen Sie die Vorlage „TwinCAT driver Project“ und geben einen Treibernamen ein, „StateMachineDrv“ in diesem Beispiel. Klicken Sie auf „Add“ um fortzufahren.

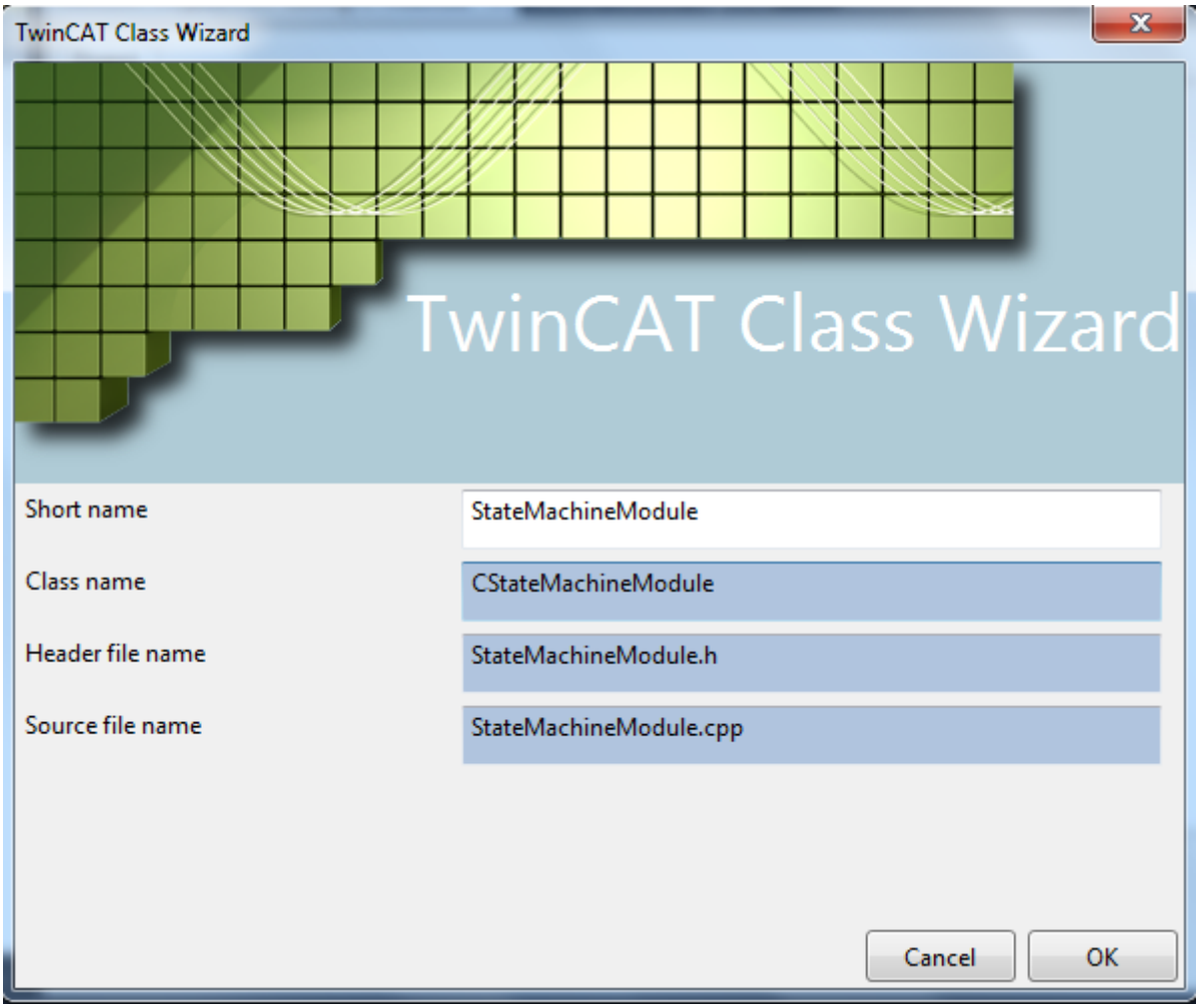


3. Wählen Sie eine für diesen neuen Treiber zu verwendende Vorlage. In diesem Beispiel wählten wir den „TwinCAT Module Class with Cyclic IO“ da der interne Zähler der Zustandsmaschine verfügbar ist, um den IO zugeordnet zu werden.
4. Klicken Sie auf „Add“ um fortzufahren.


Add New Item - StateMachineDrv
Installed Templates
TwinCAT C++ Module
Online Templates
Sort by: Default
Search Installed Templates
TwinCAT Module Class TwinCAT C++ Module
TwinCAT Module Class with ADS port TwinCAT C++ Module
TwinCAT Module Class with Cyclic Caller TwinCAT C++ Module
TwinCAT Module Class with Cyclic IO TwinCAT C++ Module
TwinCAT Module Class with Data Pointer TwinCAT C++ Module
Type: TwinCAT C++ Module
Creates a new TwinCAT module class which implements the cyclic caller interface and which has an input and output data area.
Name:
Location:
Browse...
Add
Cancel

5. Einen Namen für die neue Klasse im C++ Treiber „StateMachineDrv“ angeben.
6. Für „Short name“ geben wir in diesem Beispiel „StateMachineModule“ ein. Durch Klicken in eines der anderen Bearbeitungsfelder wird der Name in den anderen Feldern wie „Class name“ oder „Header file name“, usw. übernommen.

7. Klicken Sie auf „OK“ um fortzufahren.

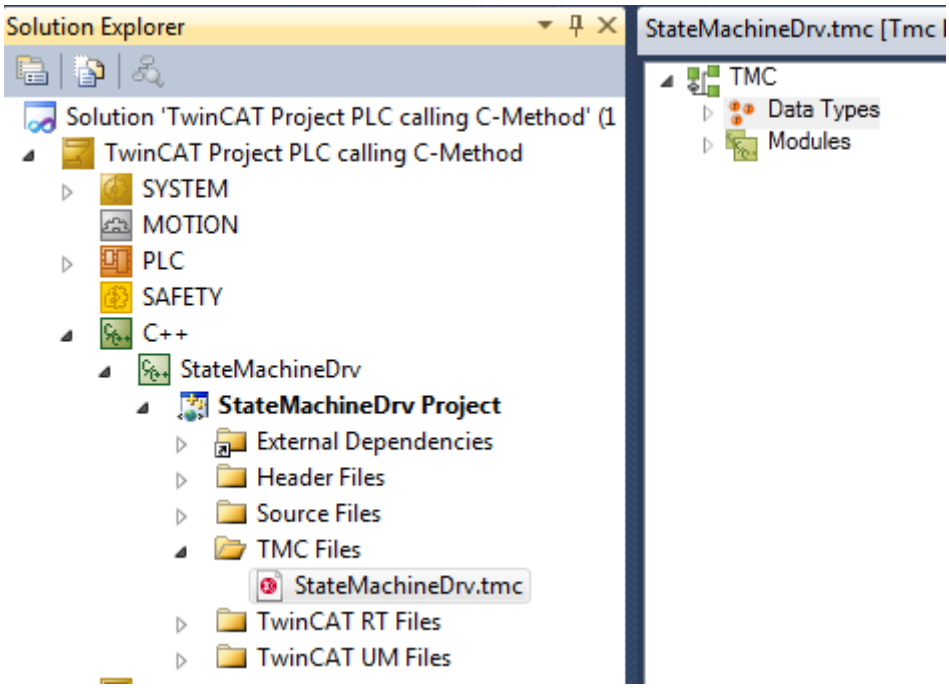


⇨ Daraufhin erstellt der Assistent ein C++ Projekt, das fehlerfrei kompiliert werden kann.

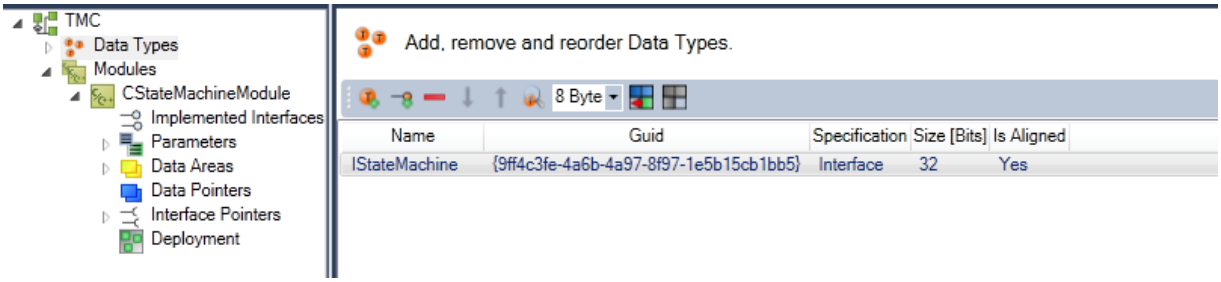
Schritt 3: Eine neue TwinCAT 3 Schnittstelle erzeugen

Hinweis	**Namenskonflikt** Bitte verwenden Sie keine der SPS vorbehaltenen Schlüsselwörter als Namen. Wenn der Treiber im Verbund mit einem SPS-Modul verwendet wird, kann es zu Namens-kollisionen kommen.

8. Starten Sie den TMC Editor mittels Doppelklick auf „StateMachineDrv.tmc“ - wählen Sie dann **„Data Types“** auf der rechten Seite.



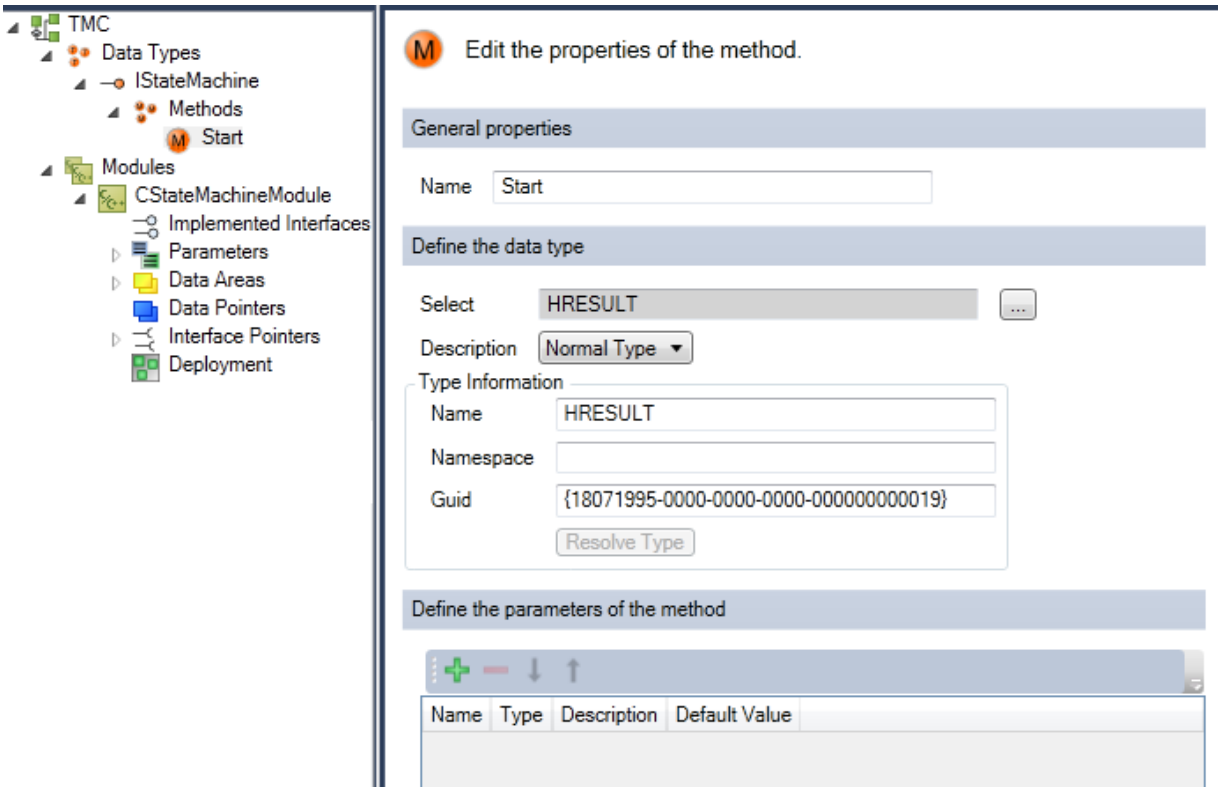
9. Fügen Sie eine neue Schnittstelle mittels Klicken auf die „+“ Taste „Add a new interface“ hinzu.
10. Daraufhin wird ein neuer Eintrag **„IInterface1“** aufgeführt, wählen Sie „IInterface1“ im linken Baum oder doppelklicken Sie auf „IInterface1“ in der Datentypenliste, um Details bezüglich der neuen Schnittstelle zu erhalten.
11. Geben Sie einen aussagekräftigeren Namen ein - in diesem Beispiel **„IStateMachine“**



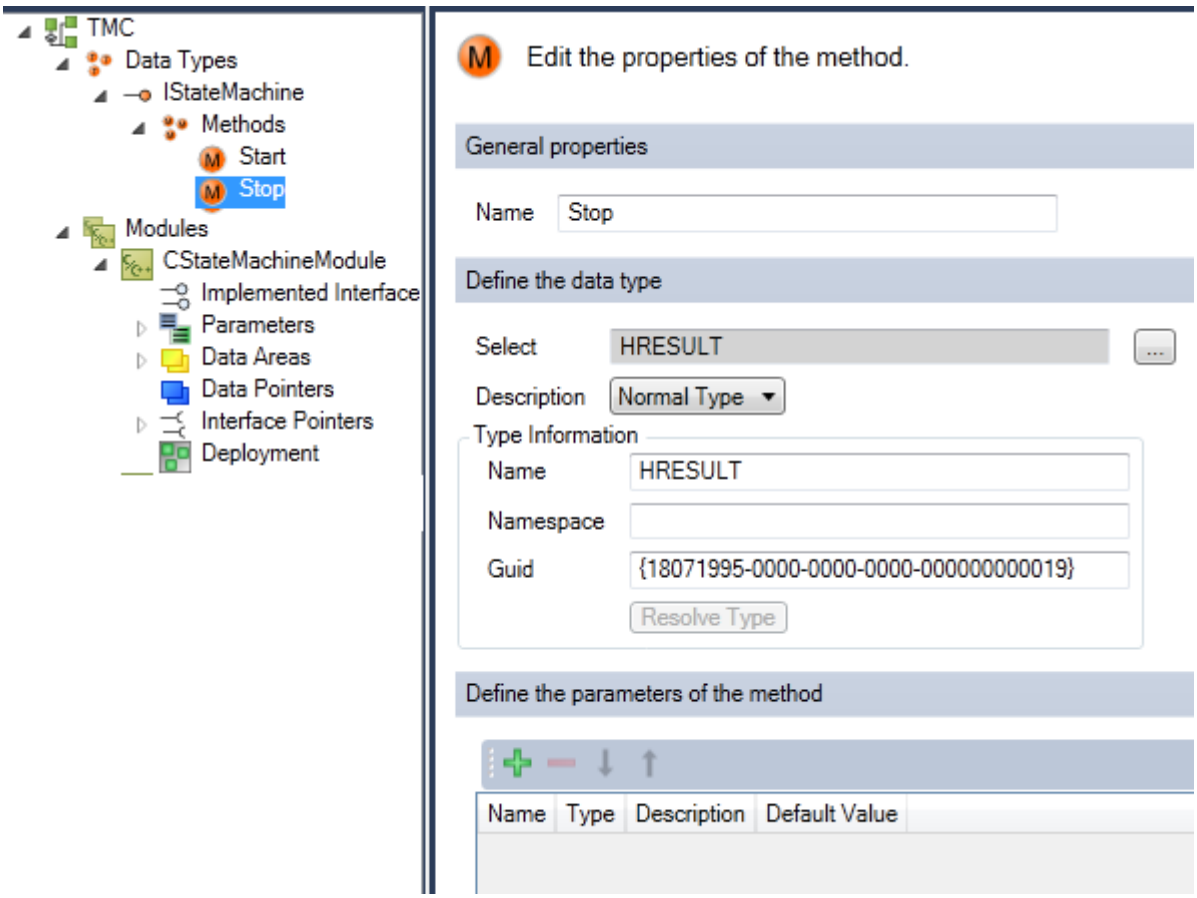
Schritt 4: Fügen Sie der Schnittstelle Methoden hinzu

12. Klicken Sie auf **„Edit Methods...“**, um eine Liste der Methoden dieser Schnittstelle zu erhalten: Klicken Sie auf die „+“ Schaltfläche um eine neue standardmäßige Methode **„Method1“** zu erzeugen:

13. Den Standardnamen „Method1“ durch aussagekräftigen Namen ersetzen, in diesem Beispiel **„Start“**.

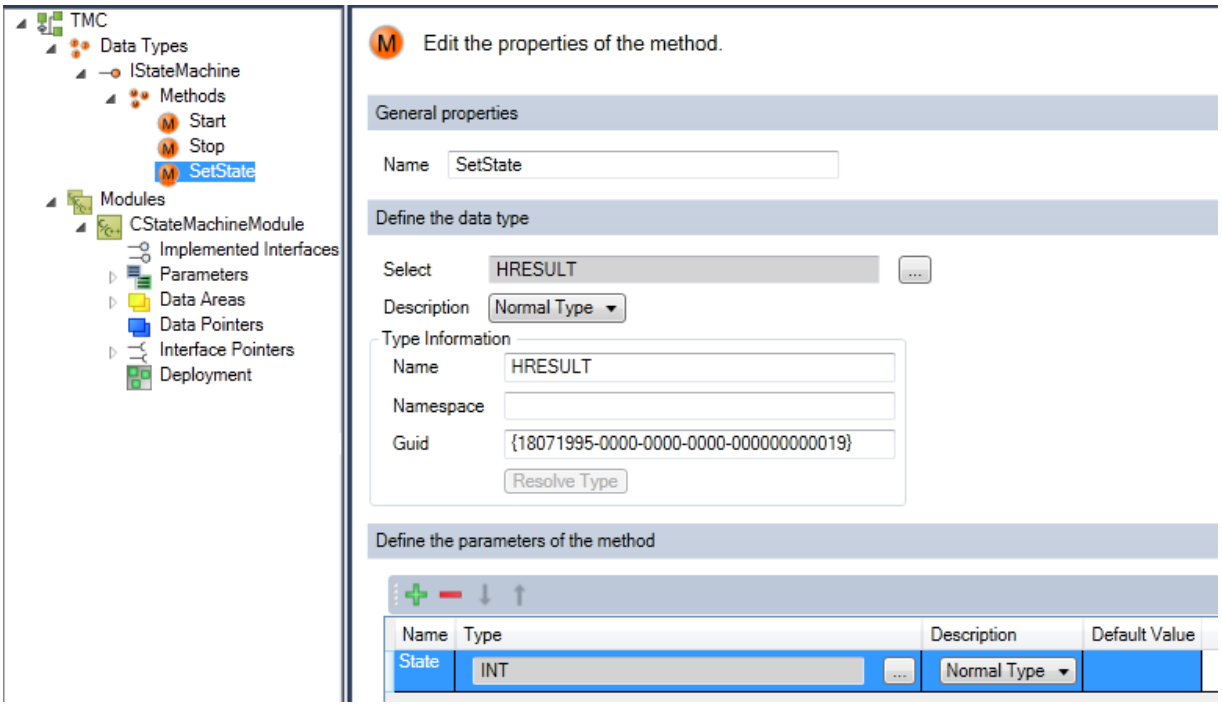


14. Ein zweite Methode hinzufügen und diese „Stop“ nennen.

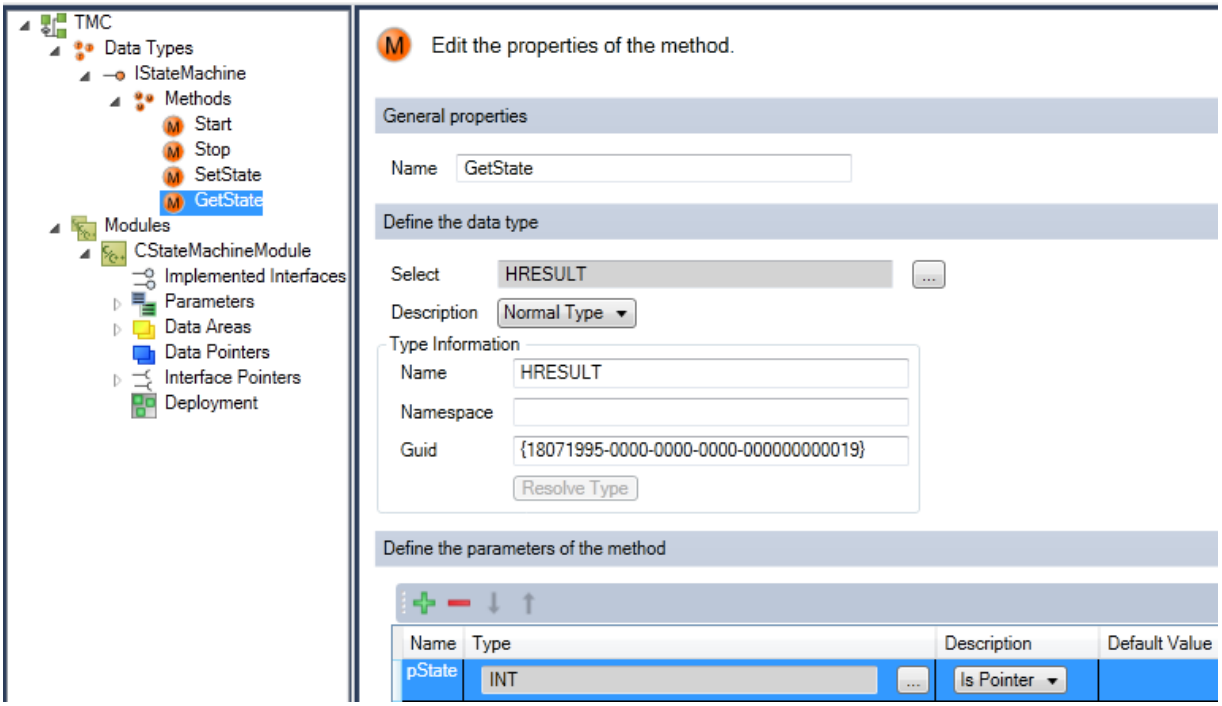


15. Ein dritte Methode hinzufügen und diese „SetState“ nennen.

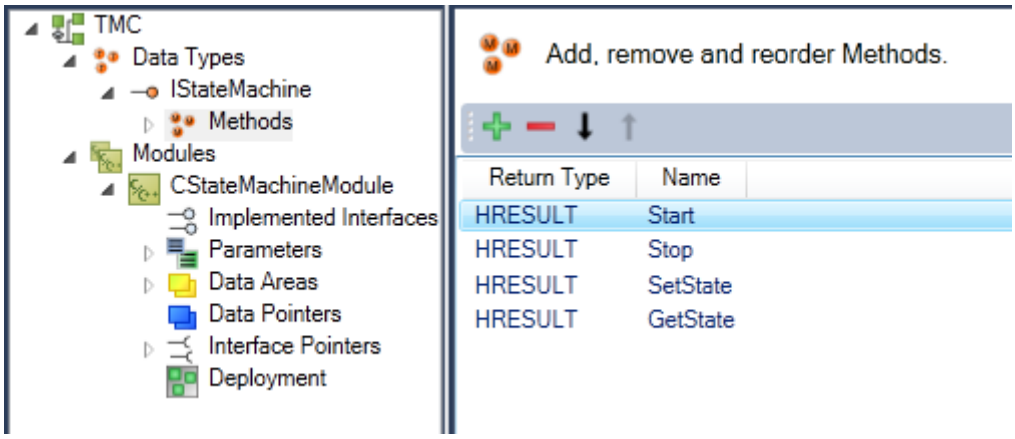
16. Anschließend können Sie mit einem Klick auf „Add a new parameter“ Parameter hinzufügen / Parameter der Methode „SetState“ bearbeiten



17. Standardmäßig wird der neue Parameter „Parameter1“ als „Normal Type“ „INTEGER“ erzeugt.
18. Auf den Namen „Parameter1“ klicken, in das Bearbeitungsfeld gehen und den Namen in „State“ ändern.
19. Nachdem „Start“, „Stop“ und „SetState“ definiert sind, eine weitere Methode definieren
20. In „GetState“ umbenennen
21. Einen Parameter hinzufügen und ihn „pState“ nennen (der so konzipiert ist, um später ein Zeiger zu werden)
22. „Normal Type“ zu „Is Pointer“ ändern

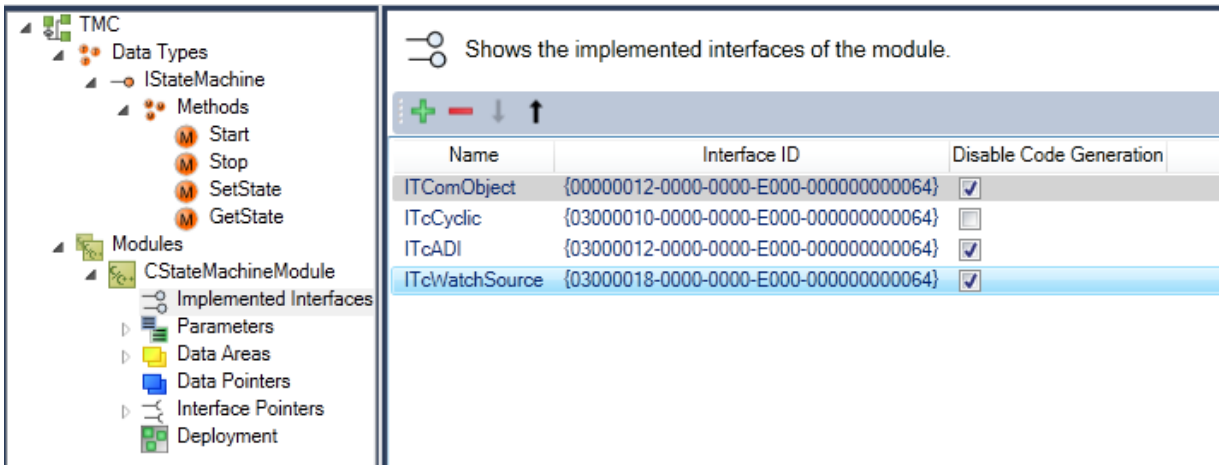


23. Daraufhin erhalten wir eine Liste aller Methoden - mit den „nach oben“ / „nach unten“ Schaltflächen kann die Reihenfolge der Methoden verändert werden.

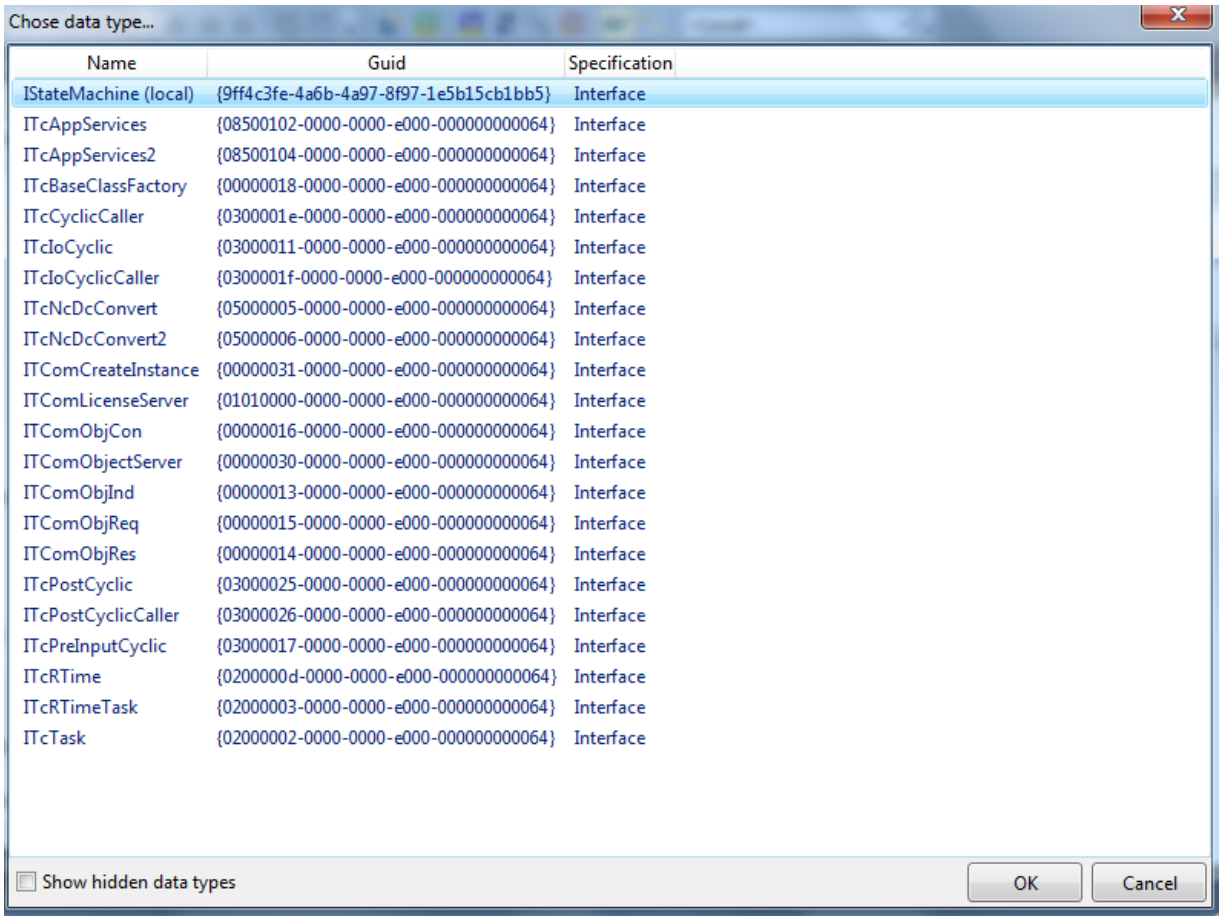


Schritt 5: Neue Schnittstelle zum Modul hinzufügen

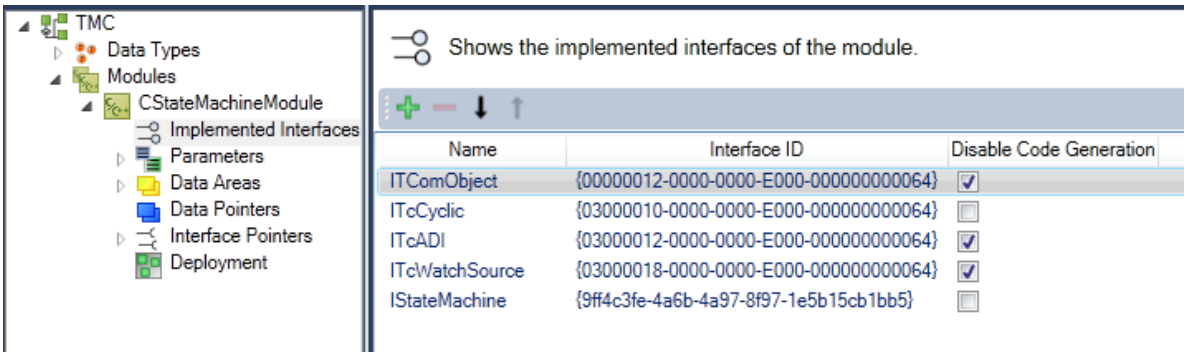
24. Wählen Sie das Modul, das mit der neuen Schnittstelle erweitert werden soll - in diesem Falle wählen Sie das Ziel **„Modules->CStateMachineModule“**.
25. Die Liste der implementierten Schnittstellen wird mittels Klicken auf die „+“ Schaltfläche mit „Add a new interface to the module“ um eine neue Schnittstelle erweitert.



26. Alle verfügbaren Schnittstellen werden aufgeführt - wählen Sie die neue Schnittstelle **„IStateMachine“** und beenden Sie mit „OK“

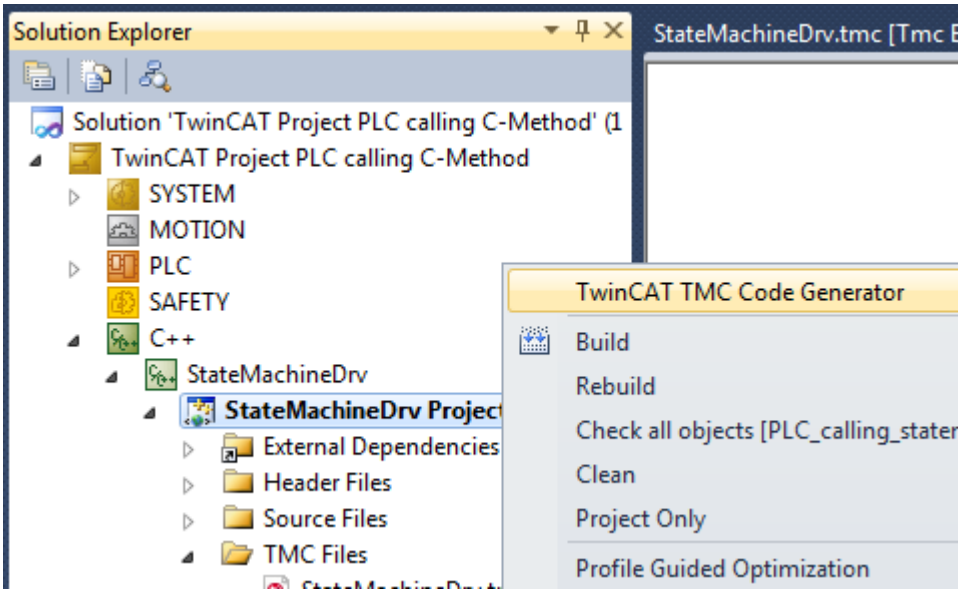


⇨ In Folge dessen ist nun die neue Schnittstelle „IStateMachine“ Teil der Modulbeschreibung.



Schritt 6: Starten Sie den TwinCAT TMC Code Generator um einen Code für die Modulklassenbeschreibung zu erzeugen

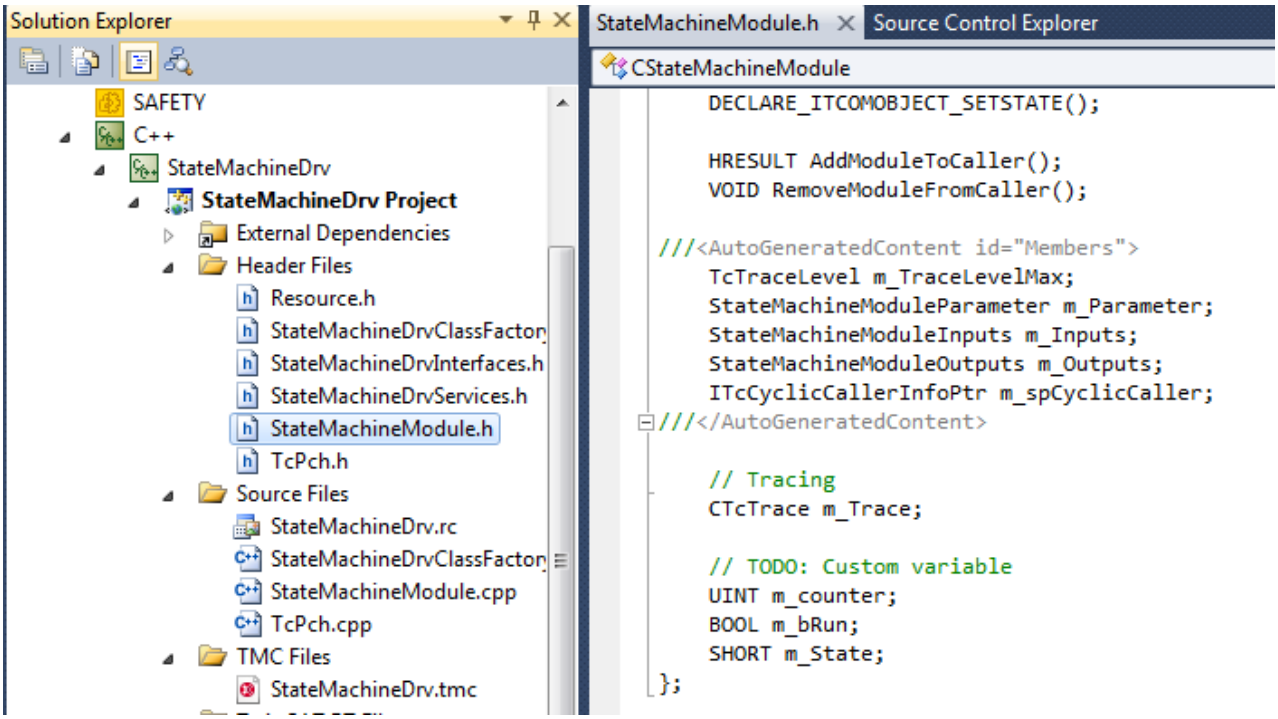
27. Um den C/C++ Code anhand von diesem Modul zu generieren, klicken Sie mit der rechten Maustaste in das C/C++ Projekt und wählen dann den **„TwinCAT TMC Code Generator“**



⇨ Daraufhin enthält das Modul **„StateMachineModule.cpp“** die neuen Schnittstellen
CModule1: Start()
CModule1: Stop()
CModule1: SetState(SHORT State)
CModule1: GetState(SHORT pState)

Schritt 7: Implementierung der Membervariablen und des Konstruktors

Membervariablen in die Header-Datei „StateMachineModule.h“ hinzufügen.

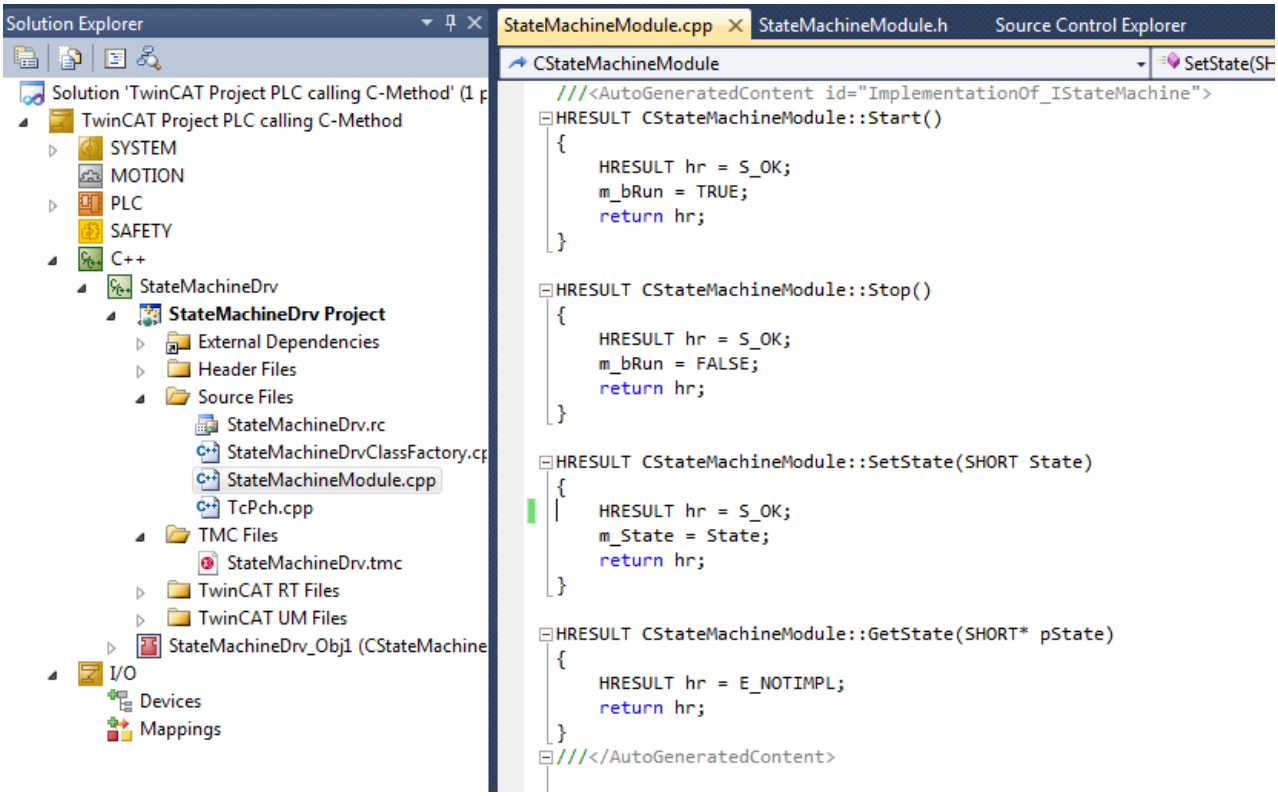


Konstruktor der Klasse füllen

```
////////////////////////////////////////////////////////////////////////
CStateMachineModule::CStateMachineModule()
    : m_Trace(m_TraceLevelMax, m_spSrv)
    , m_counter(0)
{
    memset(&m_Parameter, 0, sizeof(m_Parameter));
    memset(&m_Inputs, 0, sizeof(m_Inputs));
    memset(&m_Outputs, 0, sizeof(m_Outputs));

    m_bRun = FALSE; // by default StateMachine does not run
    m_State = 0;
}
```

Schritt 8: Methoden implementieren

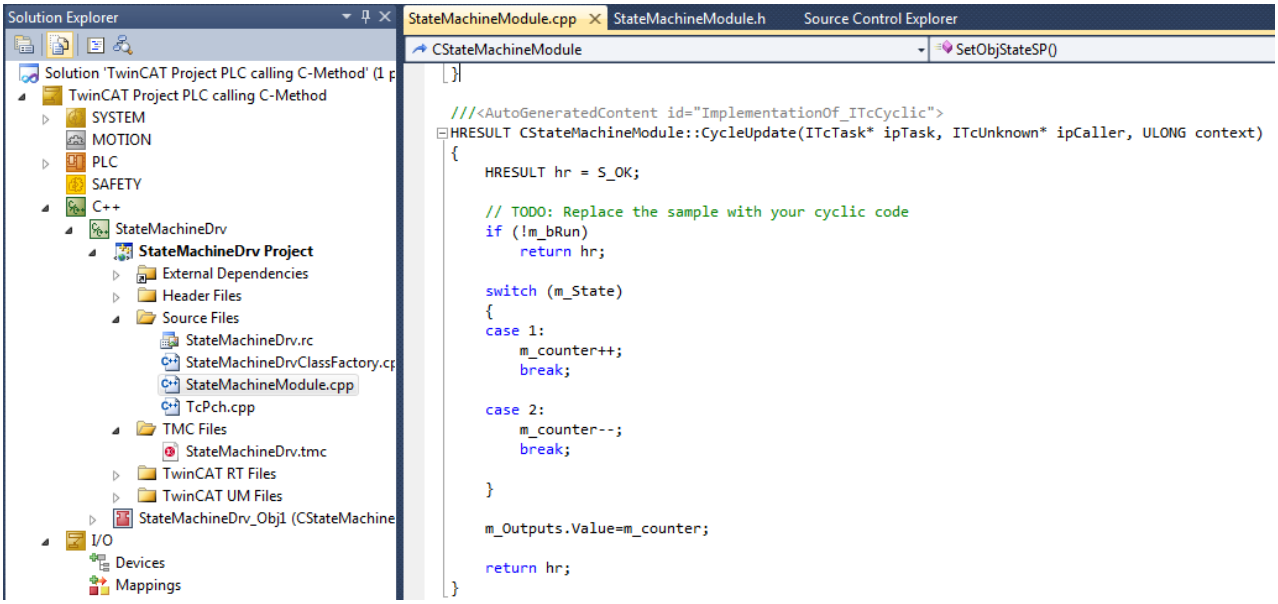
Den Code für die vier Methoden eingeben



Schritt 9: Zyklische Aktualisierung implementieren

Die C++ Modulinstanz wird zyklisch aufgerufen - selbst dann wenn die interne Zustandsmaschine sich im „Stop“ Modus befindet.

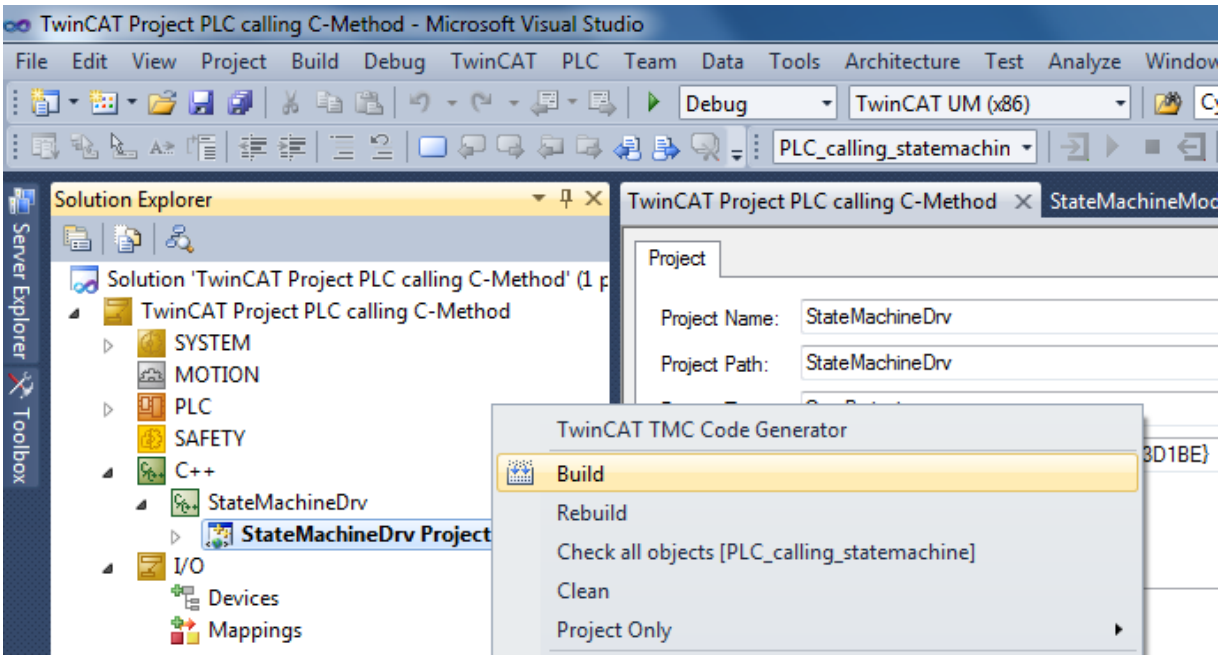
- Wenn die Zustandsmaschine nicht ausgeführt werden soll, dann signalisiert das m_bRun Flag, dass die Codeausführung der internen Zustandsmaschine zu verlassen ist.
- Bei Zustand „1“ muss der Zähler inkrementiert werden
- Bei Zustand „2“ muss der Zähler dekrementiert werden
- Der sich daraus ergebende Zählwert wird „Value“ zugewiesen, die Membervariable des logischen Ausgangs des Datenbereichs ist. Dieser kann später der physikalischen IO-Ebene oder anderen Datenbereichen von anderen Modulen zugeordnet werden.



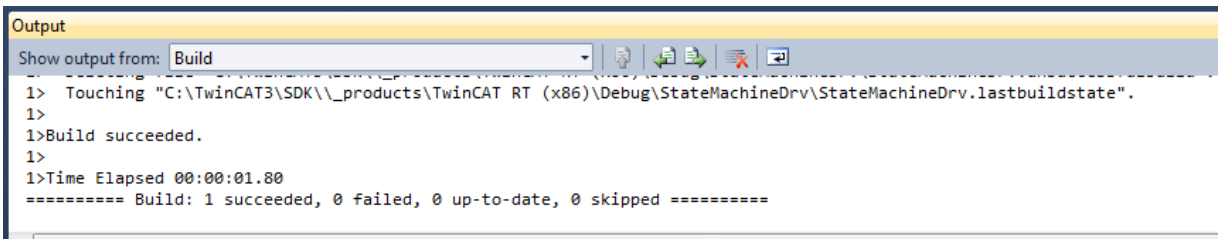
.

Schritt 10: Code kompilieren

28. Nach der Implementierung aller Schnittstellen ist es Zeit, den Code zu kompilieren - nach Rechtsklick „Build“ wählen.

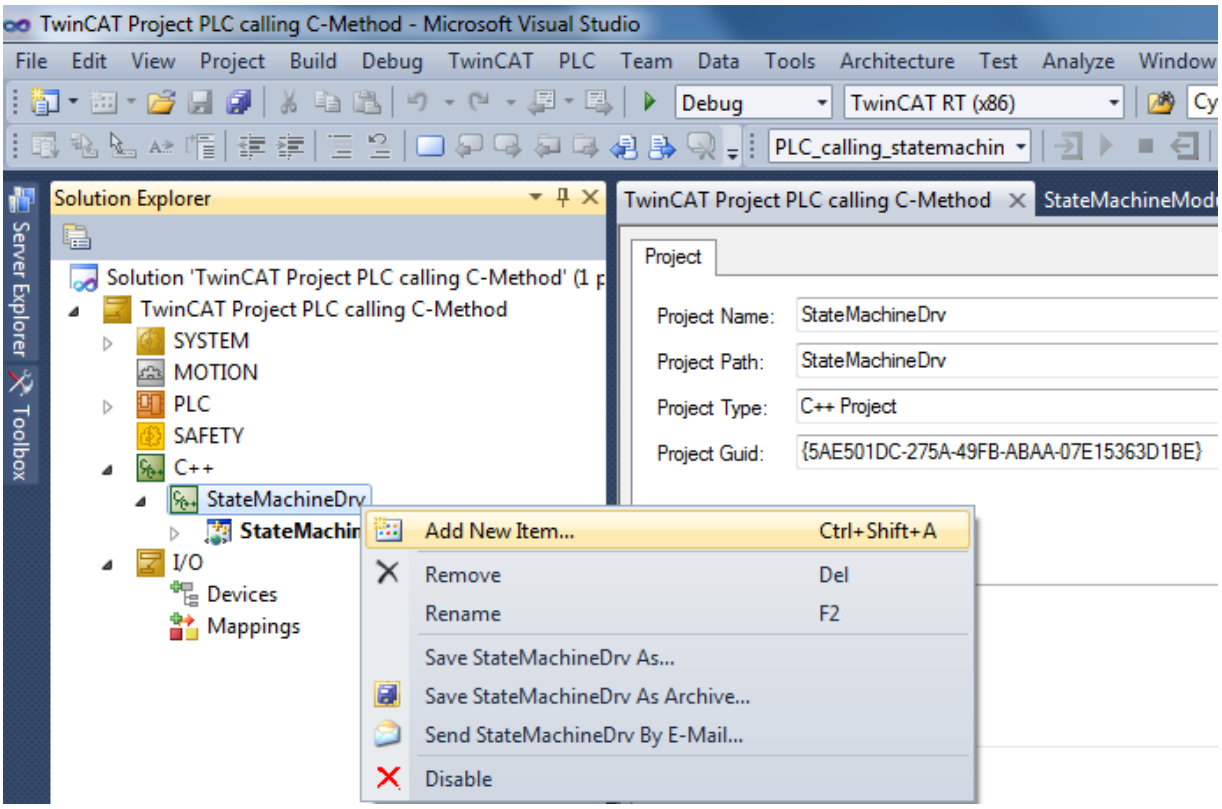


29. Die Kompilation wiederholen und Ihren Code so lange optimieren, bis das Ergebnis so aussieht.

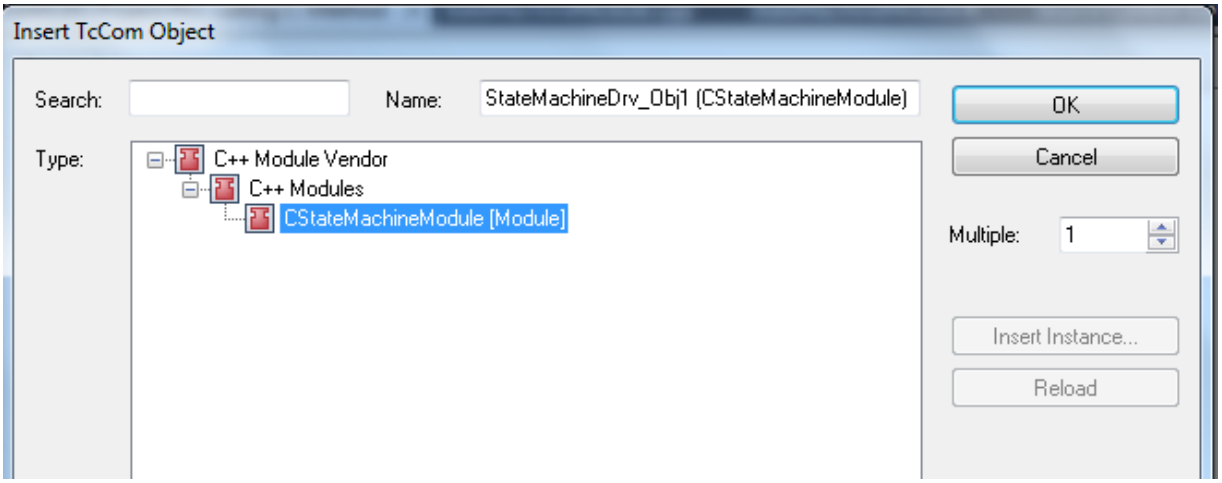


Schritt 11: Eine Instanz des C++ Moduls erstellen

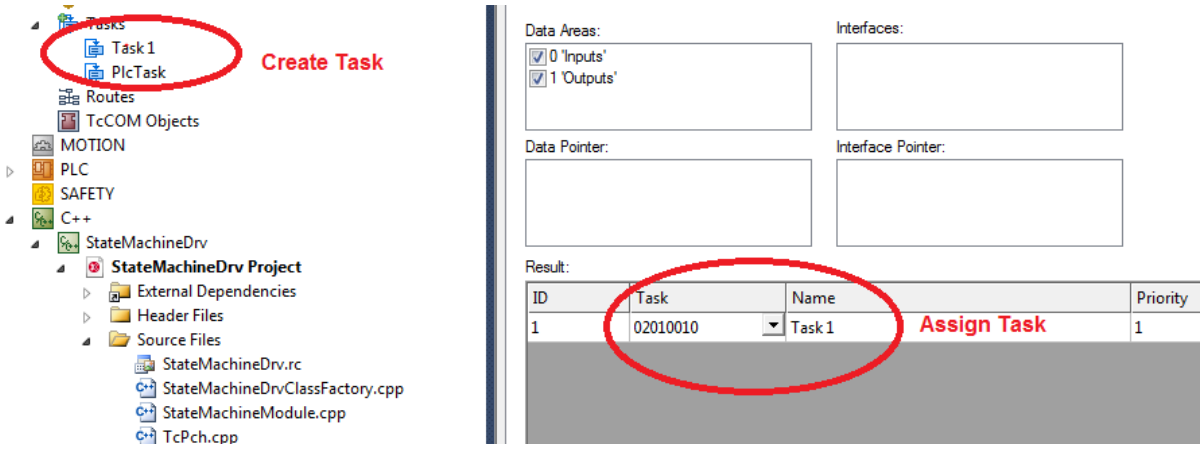
30. Rechtsklick auf das C++ Projekt und Auswahl von „Add New Item...“ um eine neue Modulinstanz zu erstellen



31. Wählen Sie das Modul aus, das als neue Instanz hinzugefügt werden soll - in diesem Falle „CStateMachineModule“



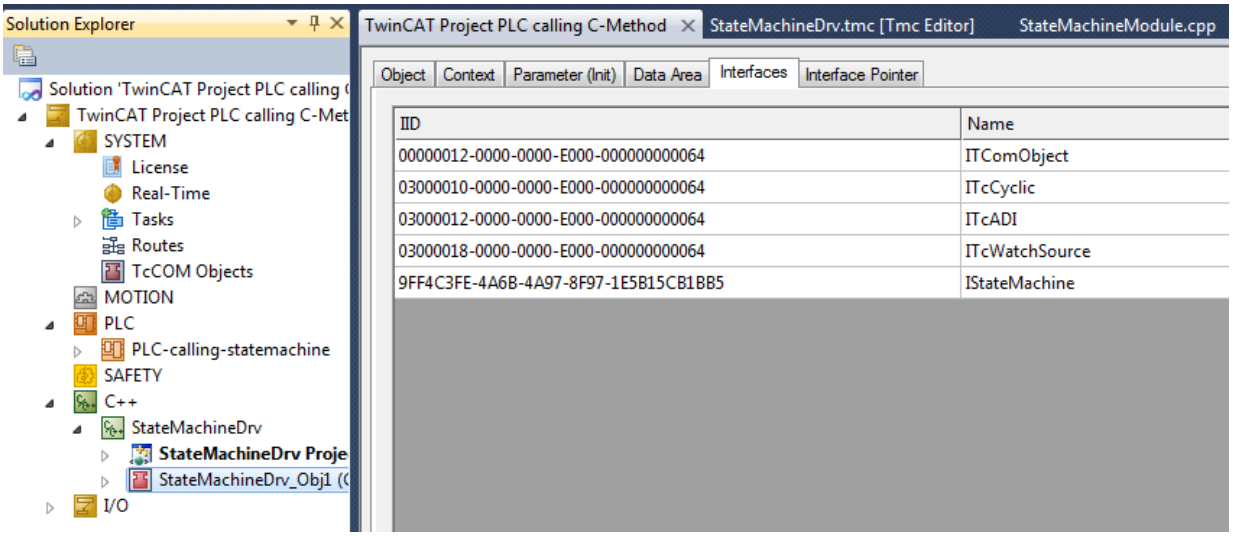
32. Wie üblich, muss die Instanz einem Task zugeordnet werden:



Schritt 12: Fertig - Ergebnis überprüfen

33. Navigieren Sie zum neuen im Solution-Baum aufgeführten Modul und wählen die Registerkarte „Interfaces“ auf der rechten Seite.

⇨ Die neue Schnittstelle „IStateMachine“ ist aufgelistet - Gut gemacht!



15.9.2 Beispiel11: SPS um eine Methode aufzurufen, die von einem anderen Modul angeboten wird (hier TwinCAT 3 C++ Modul)

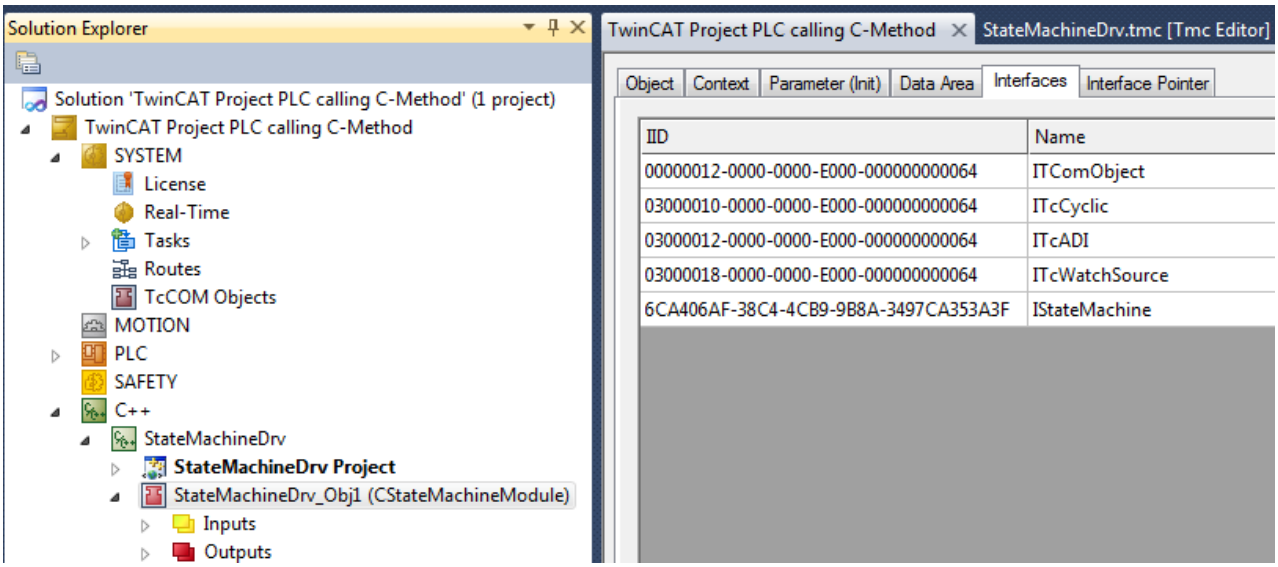
Dieser Artikel beschreibt wie eine SPS eine Methode, die von einem anderen Modul bereitgestellt wird, aufrufen kann - hier: das zuvor definierte C++ Modul.

- Schritt 1: Verfügbare Schnittstellen überprüfen [▶ 232]
- Schritt 2: Ein neues SPS-Projekt erstellen [▶ 232]
- Schritt 3: Eine neue FB-StateMachine hinzufügen [▶ 233] (die als der C++ Modulmethoden aufrufender Proxy fungiert)
- Schritt 4: Funktionsbaustein-Schnittstelle säubern [▶ 235]
- Schritt 5: FB-Schnittstellen „init“ und „FB_exit“ hinzufügen [▶ 237]
- Schritt 6: FB-Methoden implementieren [▶ 239]
- Schritt 7: FB-StateMachine in der SPS aufrufen [▶ 243]
- Schritt 8: SPS-Code kompilieren [▶ 244]
- Schritt 9: SPS FB mit C++ Instanz verknüpfen [▶ 246]

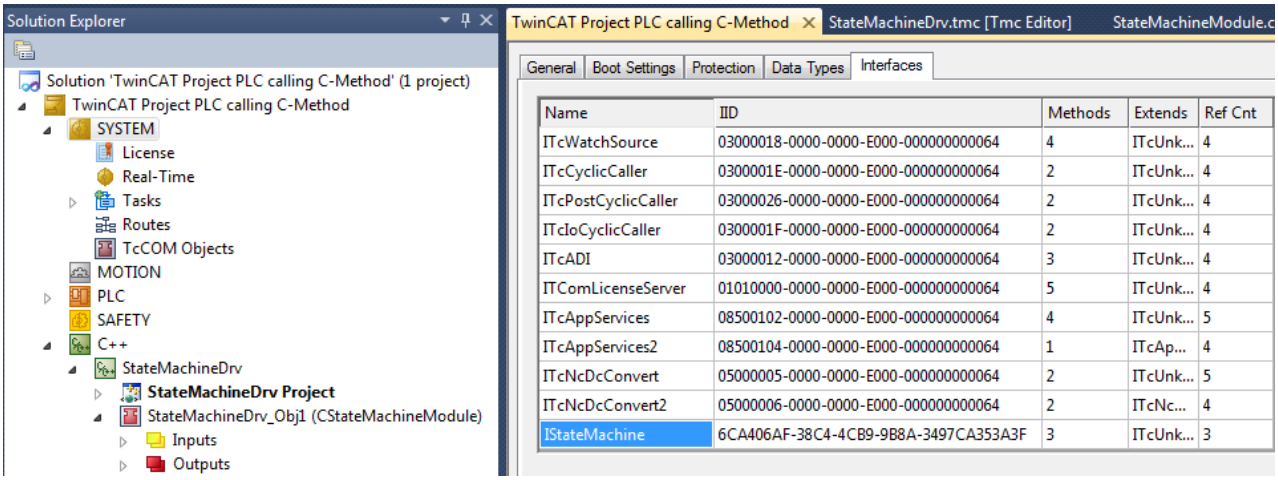
- Schritt 10: Die Ausführung von beiden Modulen, SPS und C++, beobachten [▶ 246]

Schritt 1: Verfügbare Schnittstellen überprüfen

Option 1: Zu C++ Modulinstanz navigieren, Karteireiter „Interfaces“ auswählen. Die Schnittstelle „IStateMachine“ befindet sich in der Liste, mit ihrer spezifischen IID (Interface ID)



Option 2: Zu „System“ navigieren, Karteireiter „Interfaces“ auswählen. Die Schnittstelle „IStateMachine“ befindet sich in der Liste, mit ihrer spezifischen IID (Interface ID)



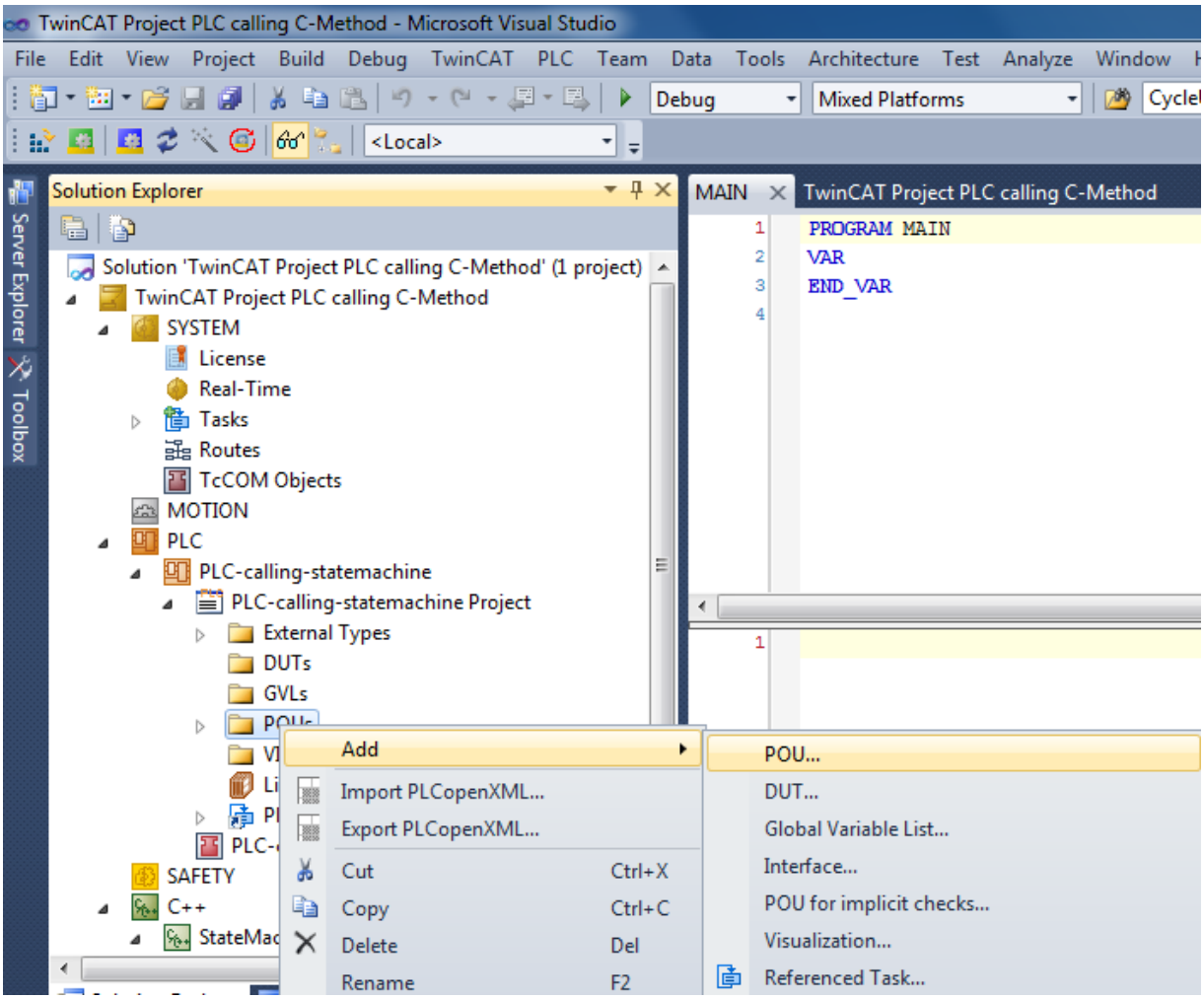
Schritt 2: Neues SPS Projekt starten

Diese Schritte sind in der TwinCAT 3 SPS Dokumentation beschrieben, siehe bitte entsprechendes Kapitel.

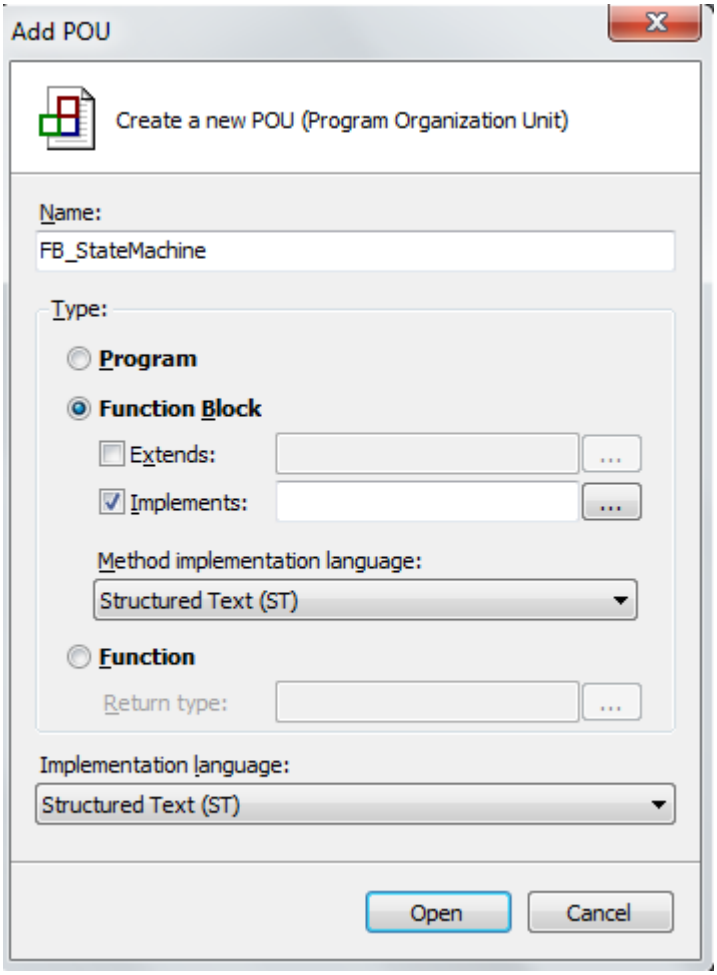
Wir erstellten hier ein SPS Projekt, genannt „PLC-calling-Statemachine“.

Schritt 3: Einen Funktionsbaustein (FB) hinzufügen (der als der C++ Modulmethoden aufrufender Proxy fungiert)

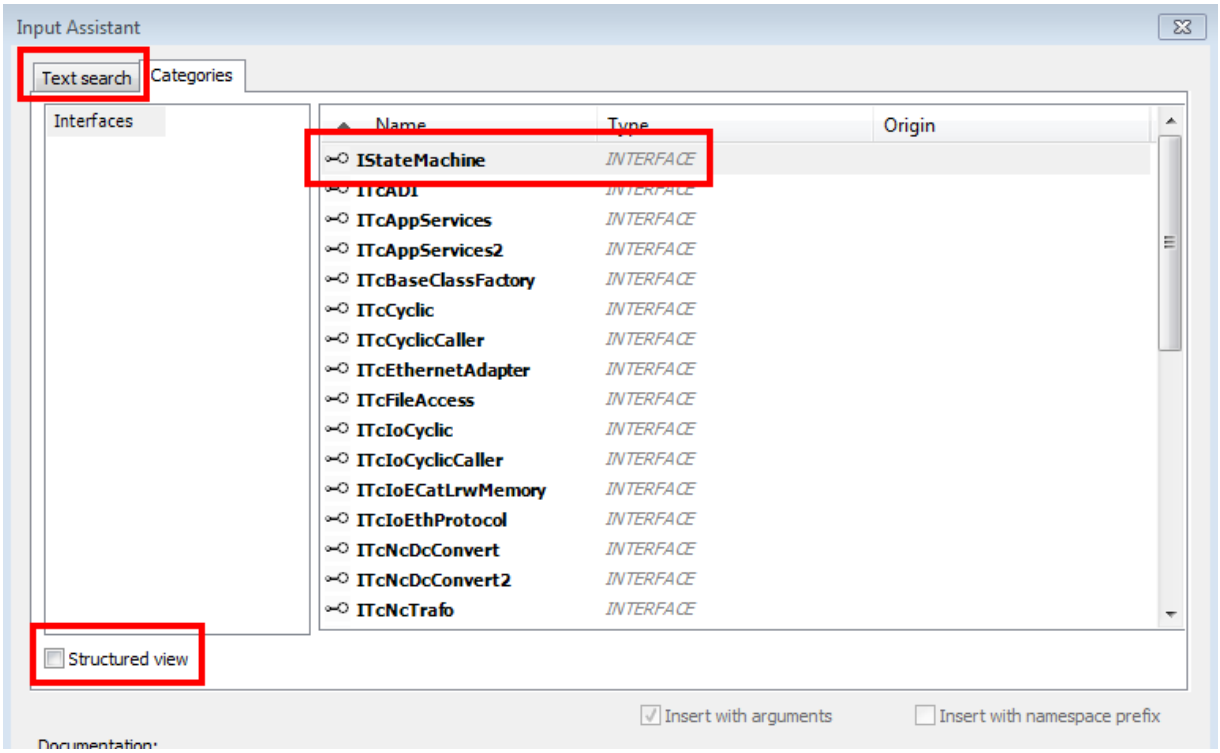
1. Auf POUs rechtsklicken und „Add->POU...“ auswählen.



2. Neuen zu erstellenden FB definieren, der später als Proxy zum Aufrufen von C++ Klasse fungiert
   Name des neuen FB eingeben: „FB_StateMachine“ und wählen Sie „Function Block“, dann „Implements“ ankreuzen und schließlich „...“ Schaltfläche auswählen

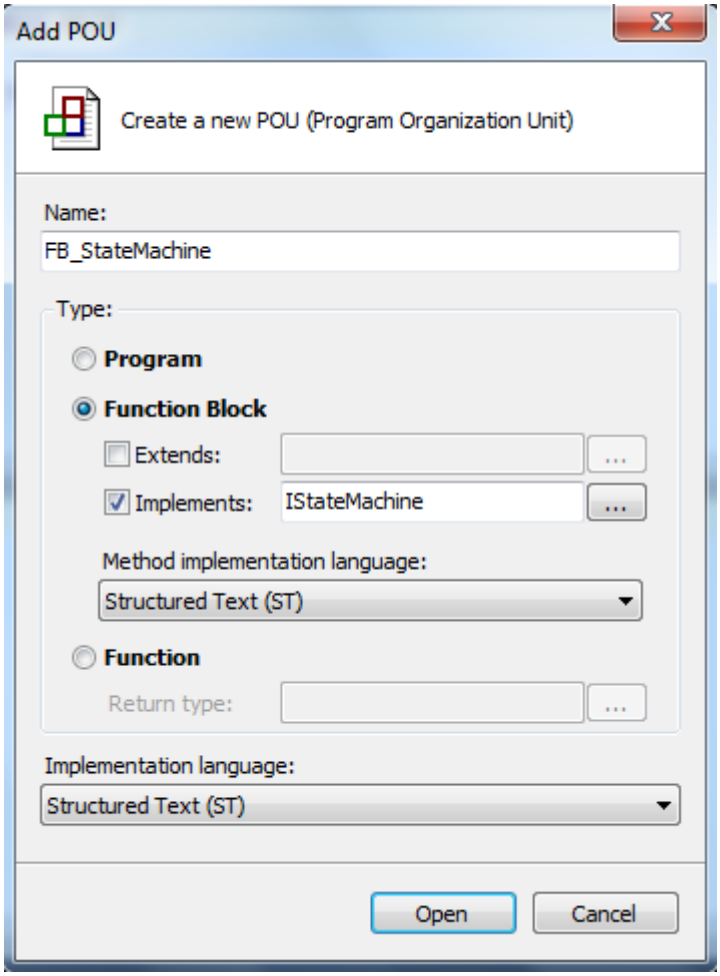


3. Die Schnittstelle entweder über „Text Search“-Karteireiter oder mittels Aufheben der Auswahl „Structured View“ bei Eingangsassistenten auswählen:



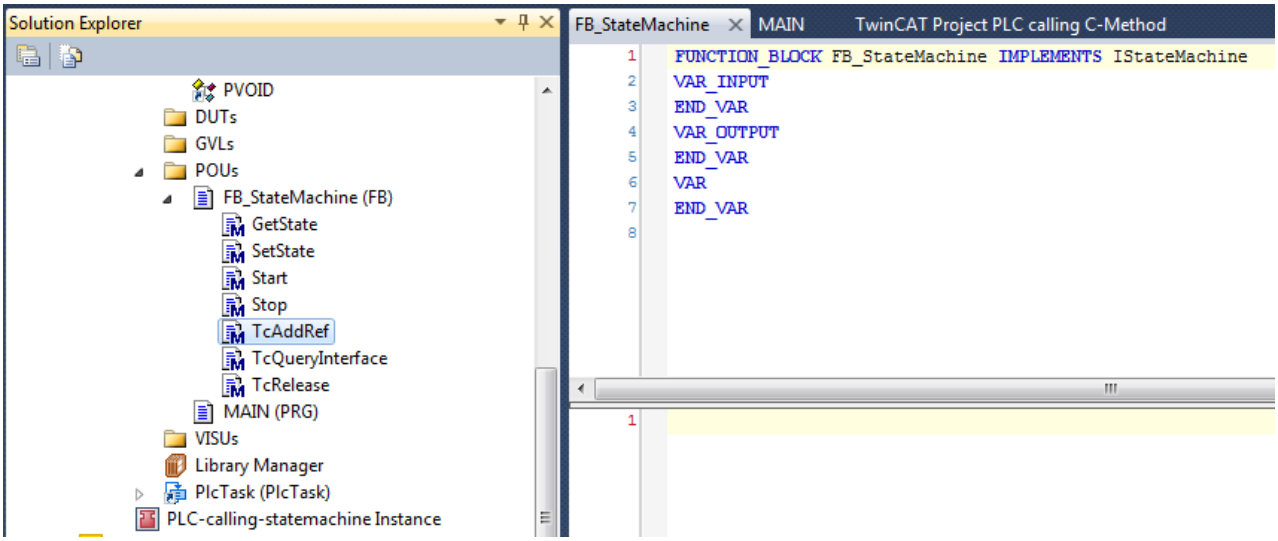
4. „IStateMachine“ auswählen und über „OK“ verlassen
   ⇨ Daraufhin wird die Schnittstelle „IStateMachine“ als zu implementierende Schnittstelle aufgelistet.

5. Wählen Sie für „Method implementation language“ „Structured Text (ST)“ aus
6. Wählen Sie für „Implementation language“ „Structured Text (ST)“ aus
7. Diesen Dialog mit „Open“ beenden



Schritt 4: Funktionsbaustein-Schnittstelle säubern

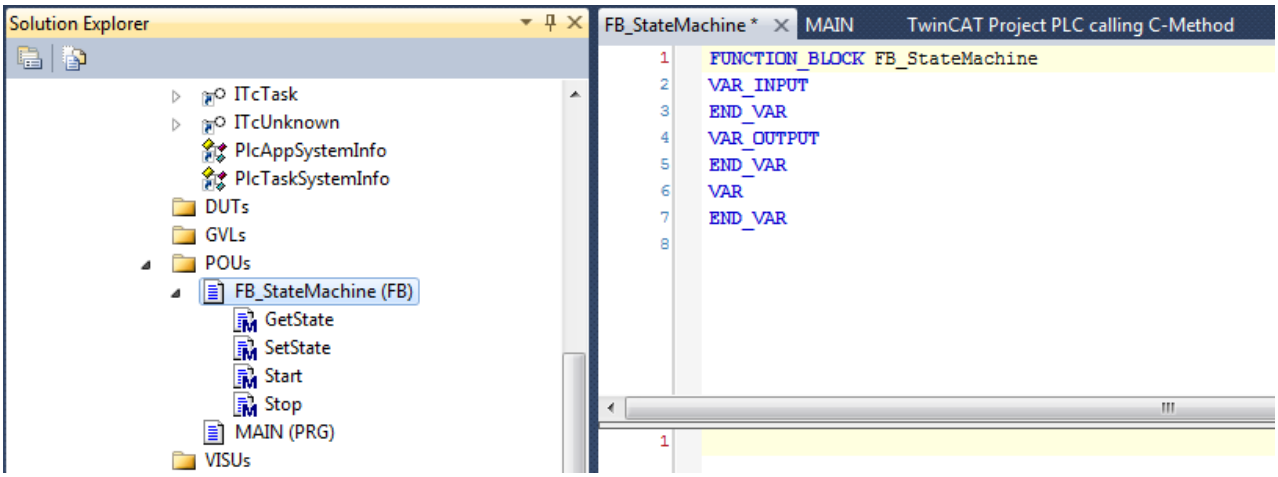
Als Ergebnis der Erstellung eines FB, der die Schnittstelle IStateMachine implementiert, wird der Assistent einen FB mit entsprechenden Methoden erstellen.



Der FB_StateMachine stellt 4 Methoden zur Verfügung:

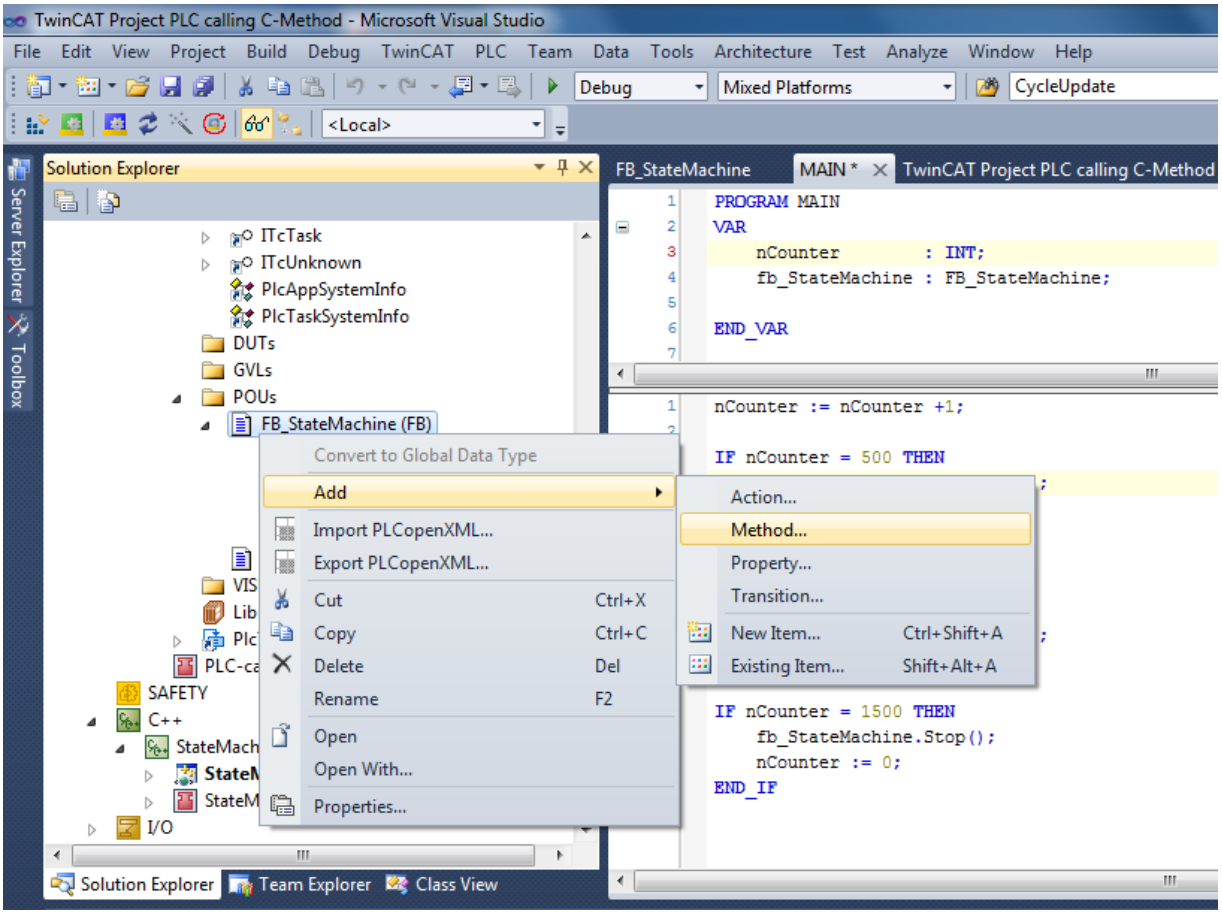
- GetState
- SetState
- Start
- Stop

Weitere erzeugte Methoden sind für dieses Beispiel ohne Belang.

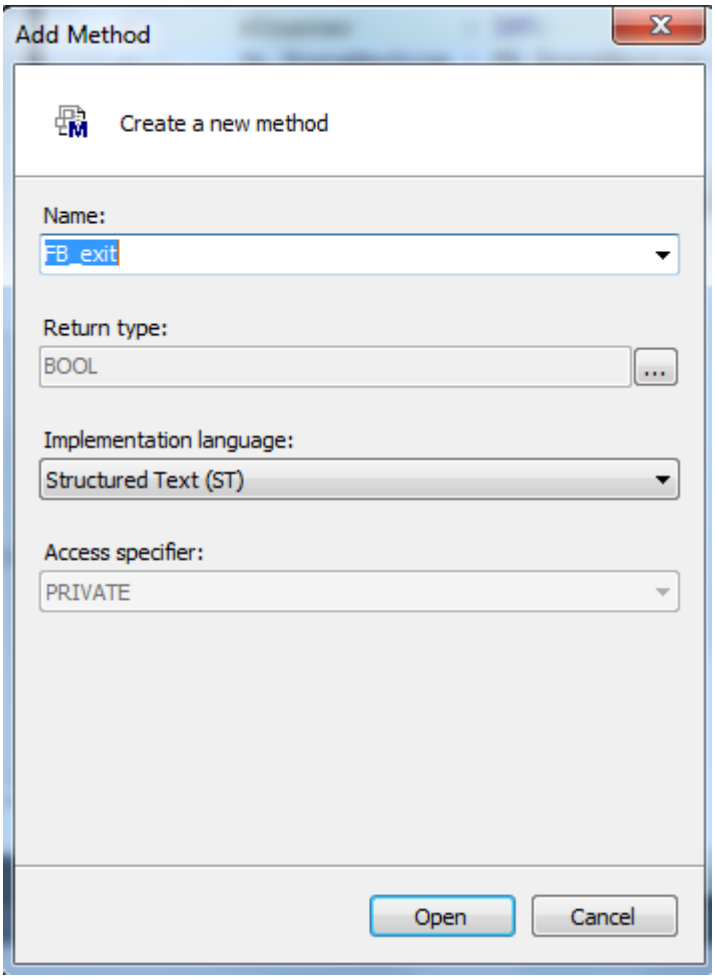


Schritt 5: FB-Schnittstellen „FB_init“ und „FB_exit“ hinzufügen

8. Rechtsklick auf „FB_StateMachine“ im Baum und „Add“ --> „Method..“ auswählen

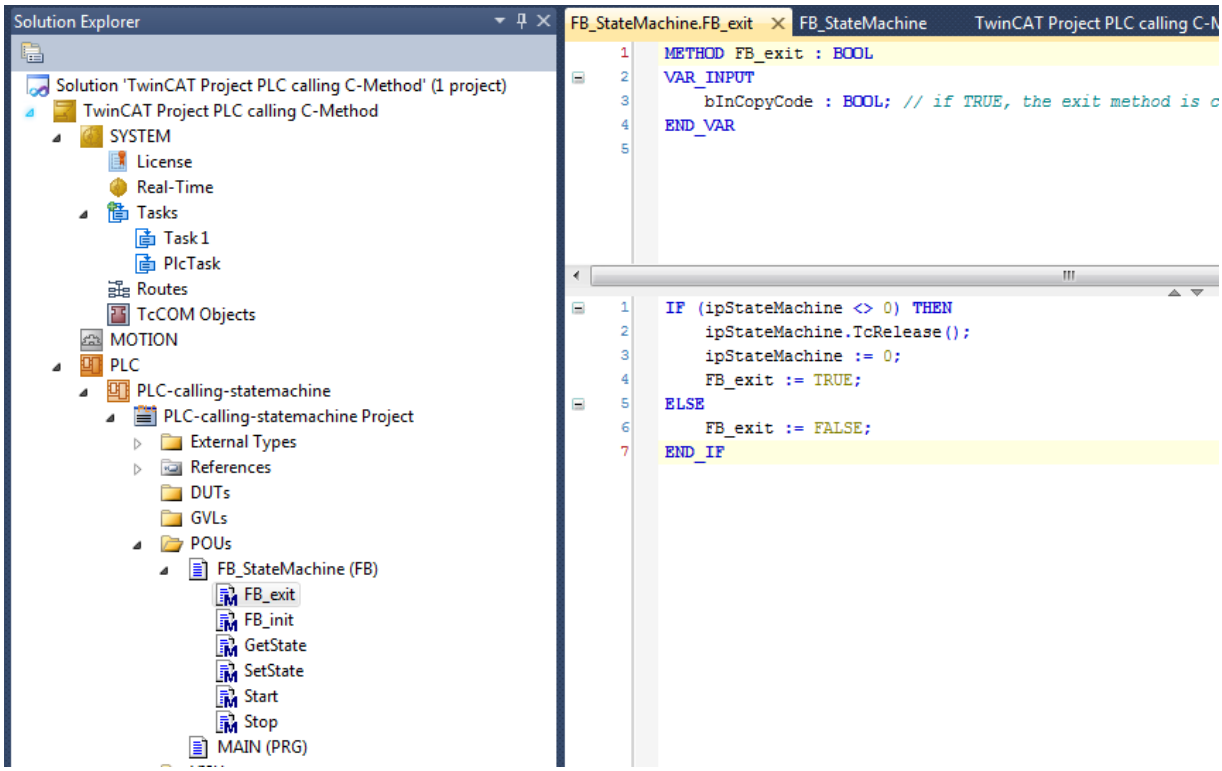


9. Fügen Sie die Methoden „FB_exit“ und „FB_init“ hinzu - beide mit „Structured Text (ST)“ als „Implementation language“.



10. Dialog mittels Klicken auf „Open“ verlassen

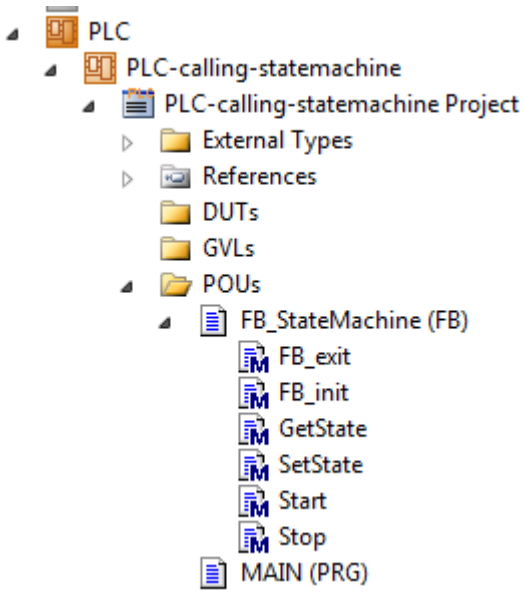
11. Rechtsklick auf „FB_StateMachine“ im Baum und „Add“ --> „Method..“ auswählen um die „FB_init“ Methode hinzuzufügen.



12. Wählen Sie wiederum für „Implementation language“ „Structured Text (ST)“ aus.
13. Dialog mittels Klicken auf „Open“ verlassen.

HINWEIS! TwinCAT 3.0 hat eine leicht abweichende Vorgehensweise für die Initialisierung des FB - siehe Beispieldownload.

⇨ Zum Schluss stehen alle notwendigen Methoden zur Verfügung - das sieht so aus:



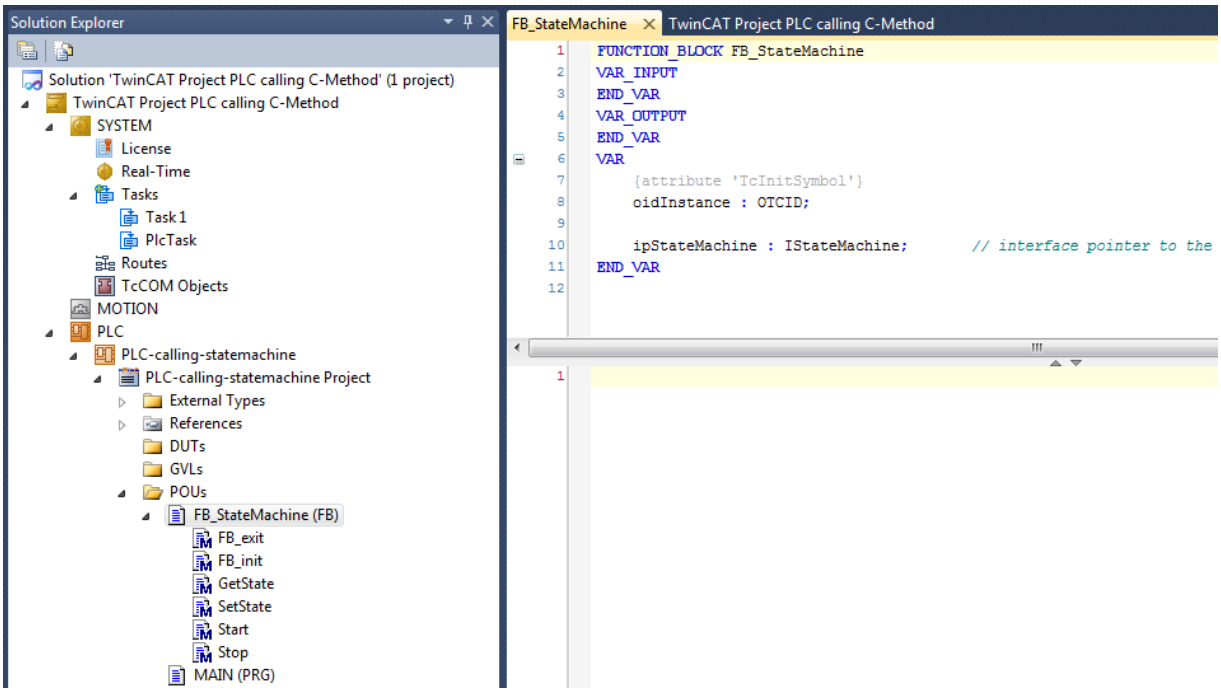
Schritt 6: FB-Methoden implementieren

Nun müssen alle Methoden mit Code gefüllt werden.

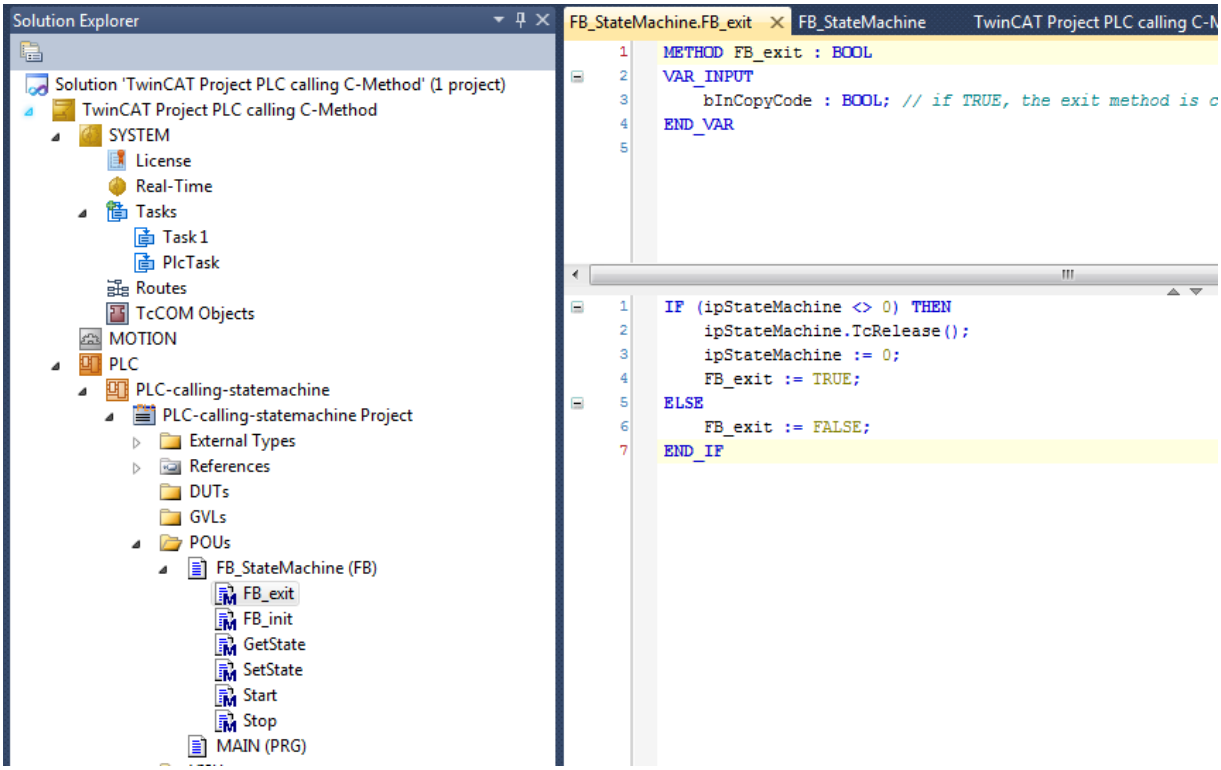
Hinweis	**Fehlende Attribute führen zu unerwartetem Verhalten** Attribut-Anweisungen mit geschwungenen Klammern stellen Code dar, der einzufügen ist.

Genaueres siehe hier.

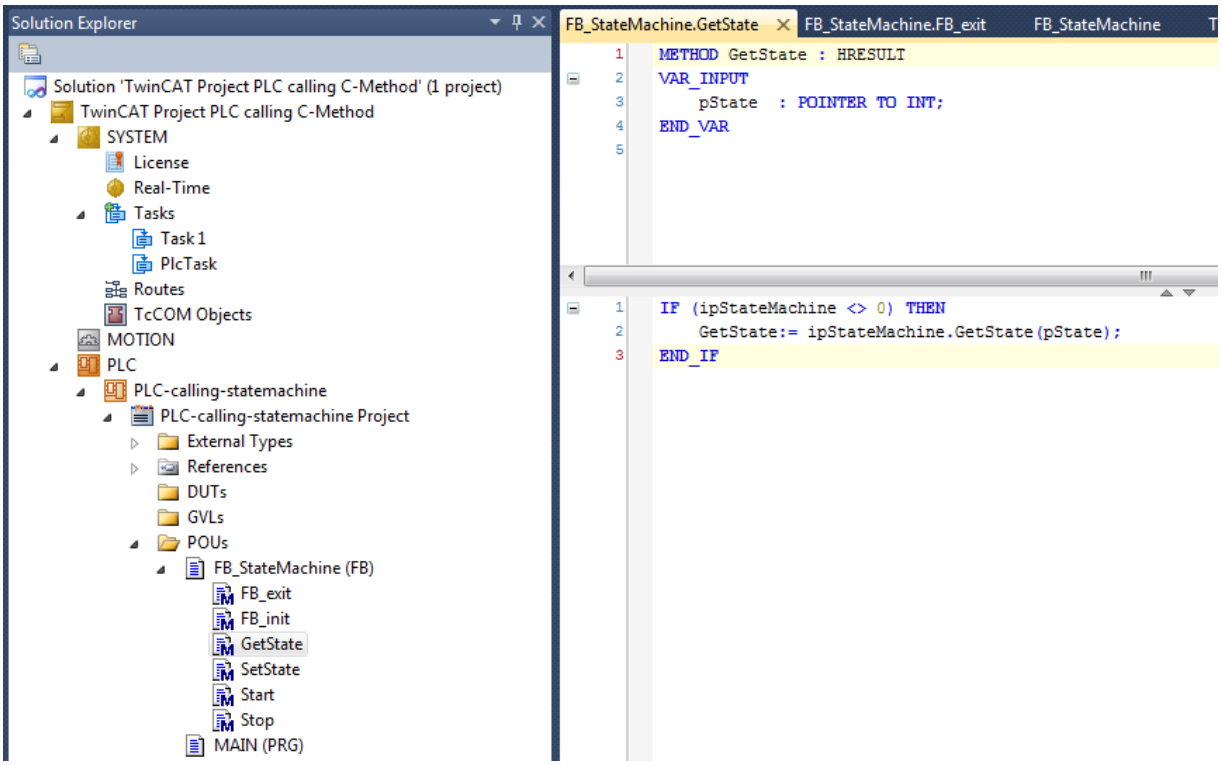
1. Variablendeklaration der FB_Statemachine implementieren:



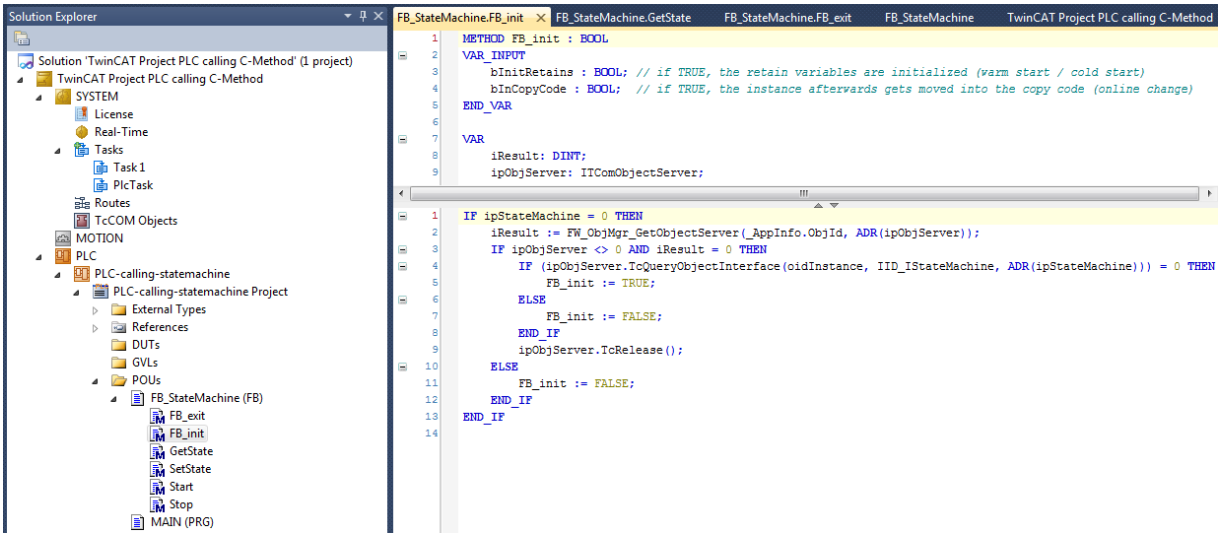
2. Variablendeklaration und Code-Bereich der Methode „FB_exit“ implementieren



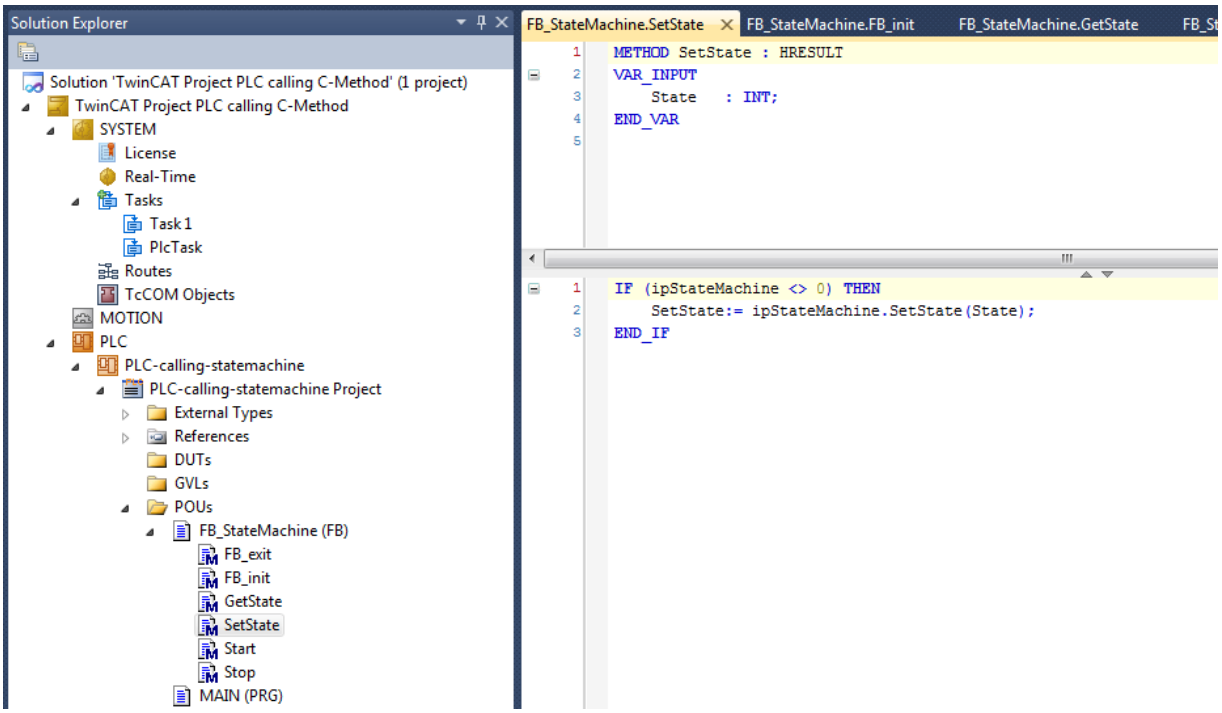
3. Variablendeklaration und Code-Bereich der Methode „GetState“ implementieren



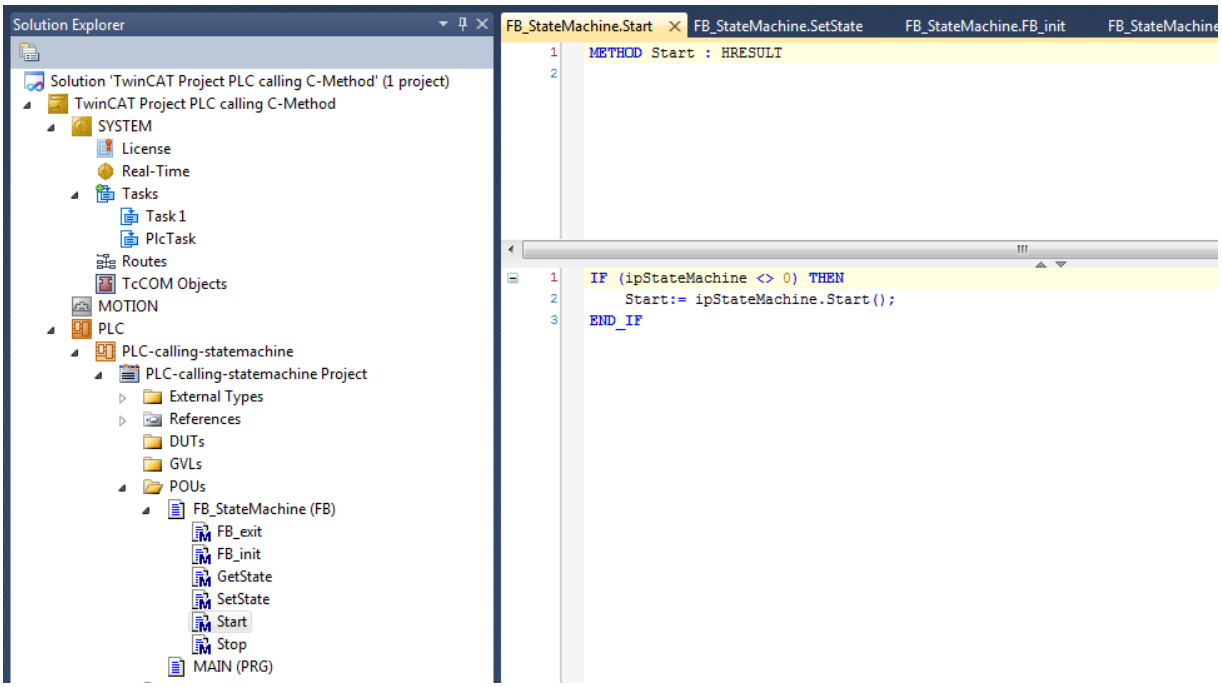
4. Variablendeklaration und Code-Bereich der Methode „FB_init“ implementieren



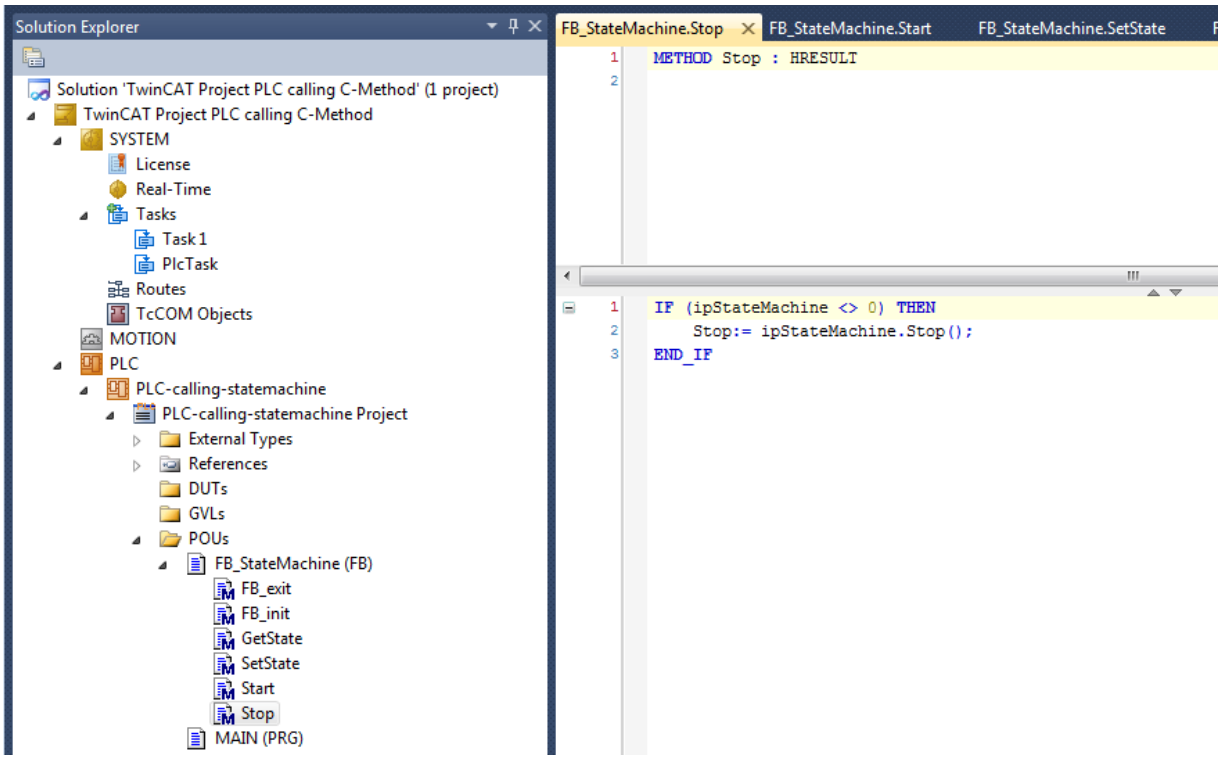
5. Variablendeklaration und Code-Bereich der Methode „SetState“ implementieren



6. Variablendeklaration und Code-Bereich der Methode „Start“ implementieren



7. Variablendeklaration und Code-Bereich der Methode „Stop“ implementieren



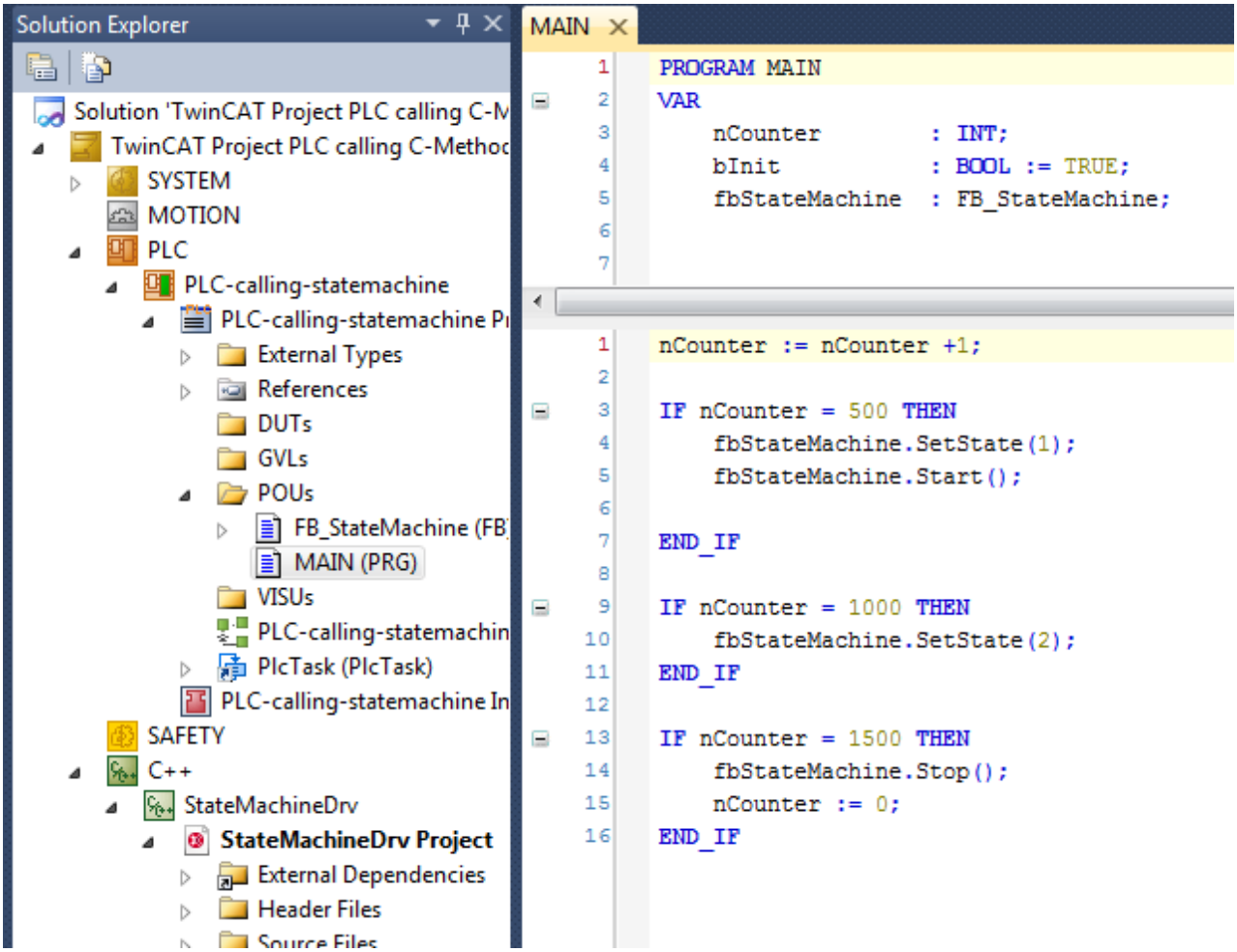
Schritt 7: FB in SPS aufrufen

Die Implementierung des „FB_StateMachine“ der als Proxy zum Aufruf der C++ Modulinstanz agiert, ist abgeschlossen.

Jetzt wird der „FB_StateMachine“ in der SPS POU „MAIN“ aufgerufen.

Dieses einfache Beispiel agiert folgendermaßen

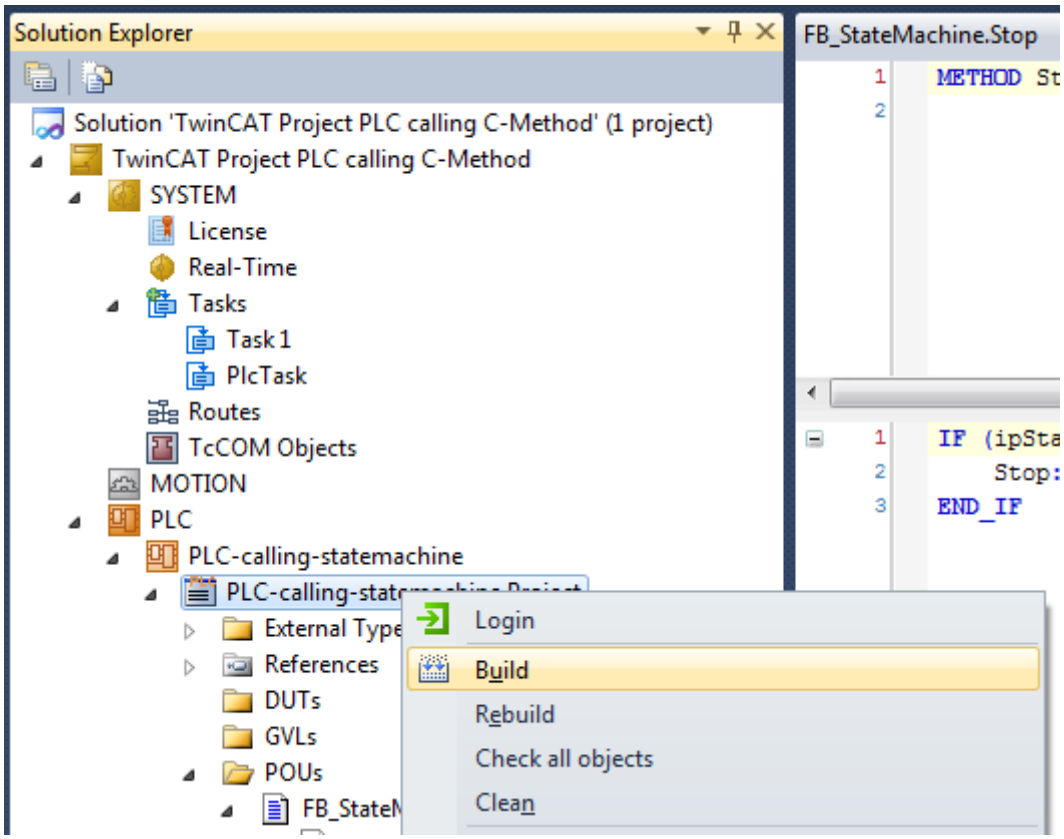
- Zyklische Inkrementierung eines SPS-Zählers „nCounter“
- Wenn nCounter = 500, dann wird die C++-StateMachine mit Zustand „1“ gestartet, um seinen internen C++ Zähler zu inkrementieren
- Wenn nCounter = 1000, dann wird die C++-StateMachine auf Zustand „2“ gesetzt, um seinen internen C++ Zähler zu dekrementieren
- Wenn nCounter = 1500 dann wird die C++-StateMachine gestoppt. Der SPS nCounter wird ebenfalls auf 0 gesetzt, sodass alles von vorne beginnt.



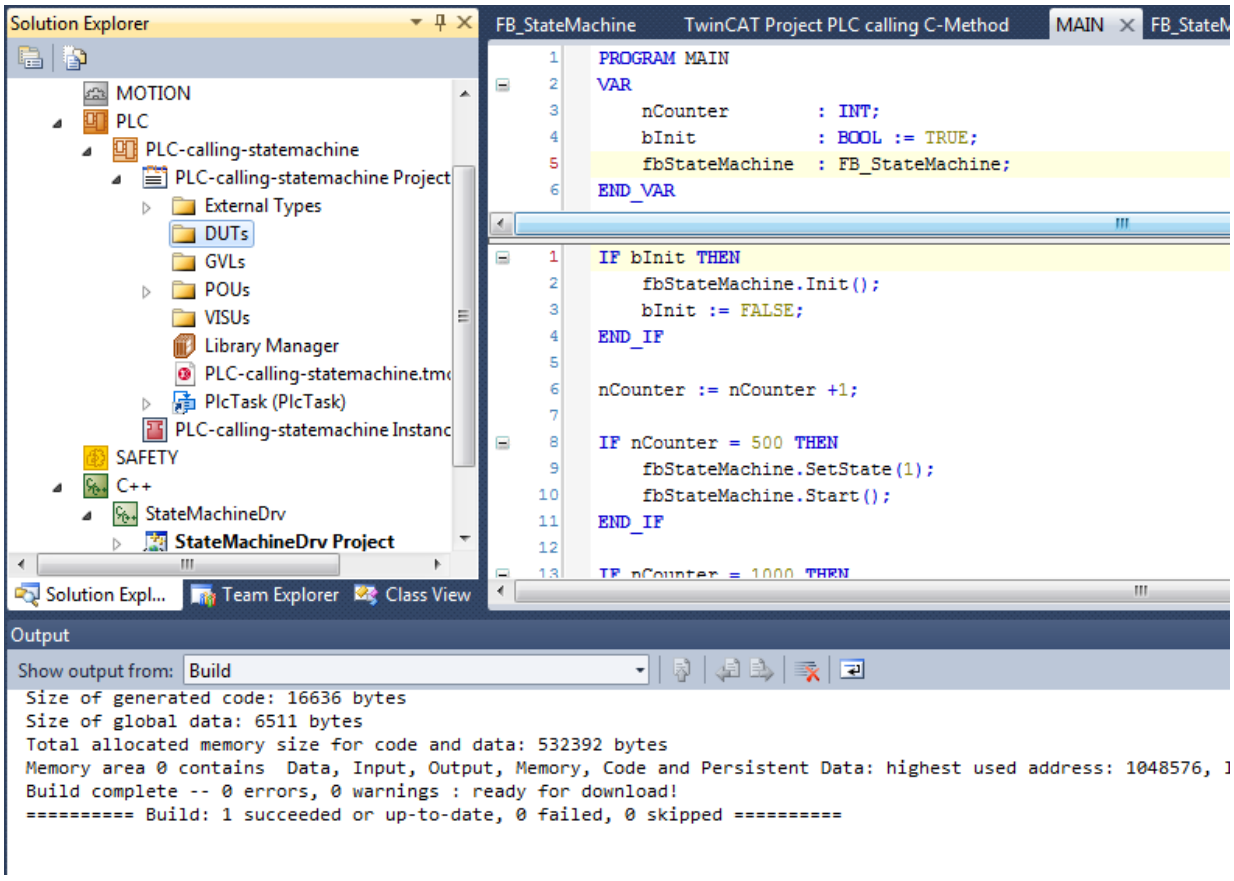
Schritt 8: SPS-Code kompilieren

Hoffentlich wurde alles korrekt implementiert. Ein guter Moment um das Projekt zu speichern und den SPS Code zu kompilieren:

8. Rechtsklick auf das SPS Projekt und Auswahl von „Build“



⇨ Gratulation - das Ergebnis der Kompilation zeigt „1 succeeded - 0 failed“.



Schritt 9: SPS FB mit C++ Instanz verknüpfen

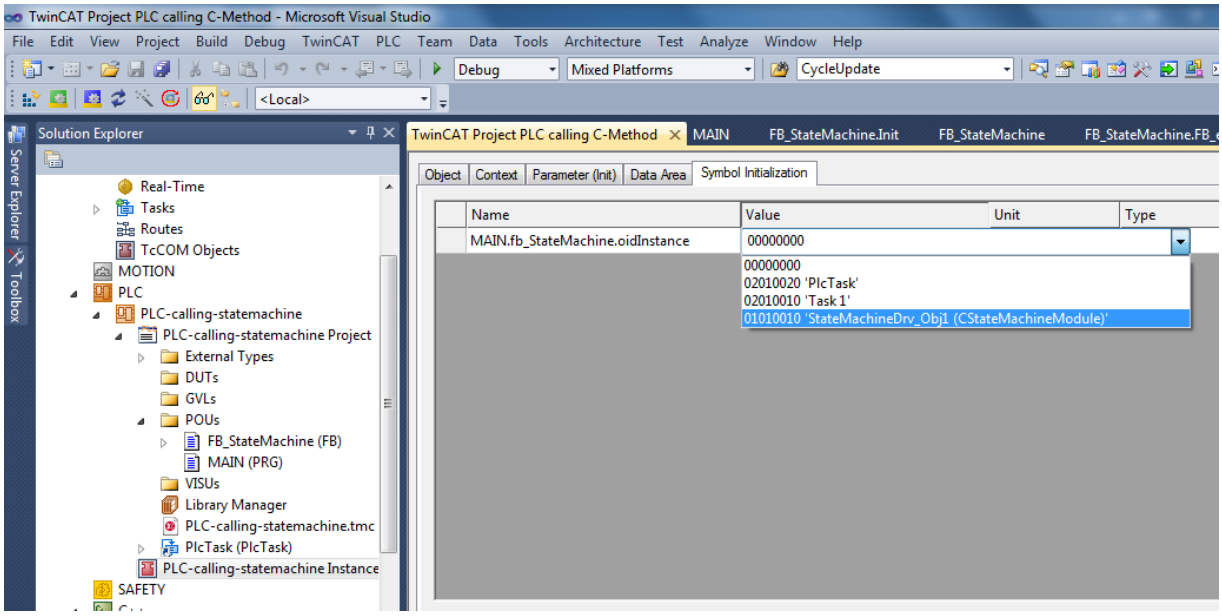
Jetzt wird man sich der Vorteile aller vorherigen Schritte bewusst:

Der SPS-FB „FB_StateMachine“ kann im Hinblick auf die Verknüpfung mit jeder Instanz des C++ Moduls „StateMachine“ konfiguriert werden. Das ist eine sehr flexible und leistungsstarke Methode, um SPS und C++ Module auf der Maschine miteinander zu verbinden.

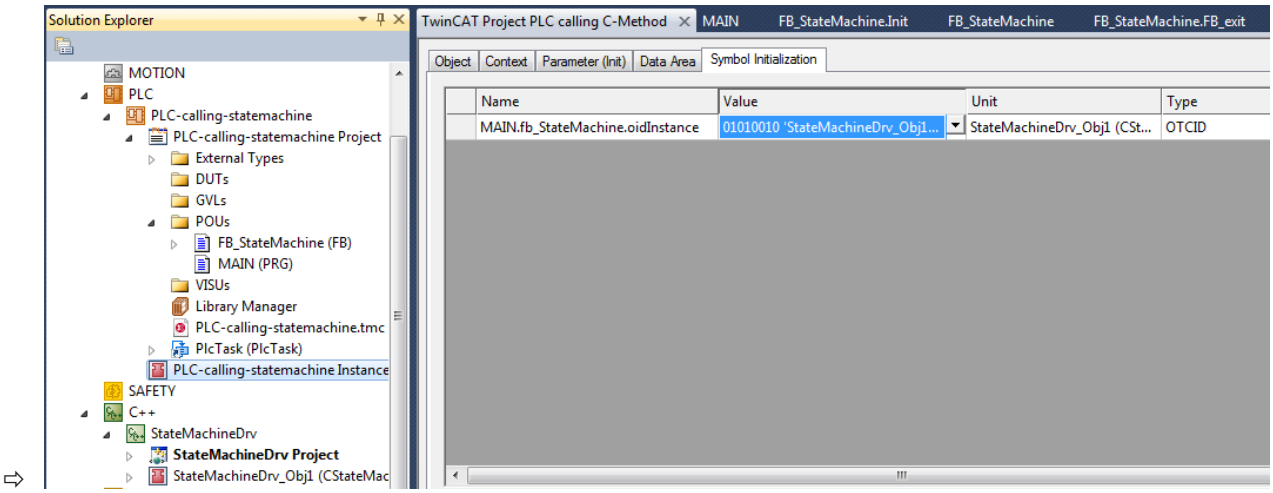
1. Navigieren Sie zur Instanz des SPS Moduls im linken Baum und wählen die Registerkarte „Symbol initialization“ auf der rechten Seite.

   ⇨ Es werden alle Instanzen von FB_StateMachine aufgelistet, in diesem Beispiel haben wir lediglich eine FB Instanz in POU „MAIN“ definiert.

2. Auswahl des Dropdownfelds „Value“ und Auswahl der C++ Modulinstanz, die mit der FB Instanz zu verknüpfen ist.



⇨ Das Ergebnis sieht dann so aus

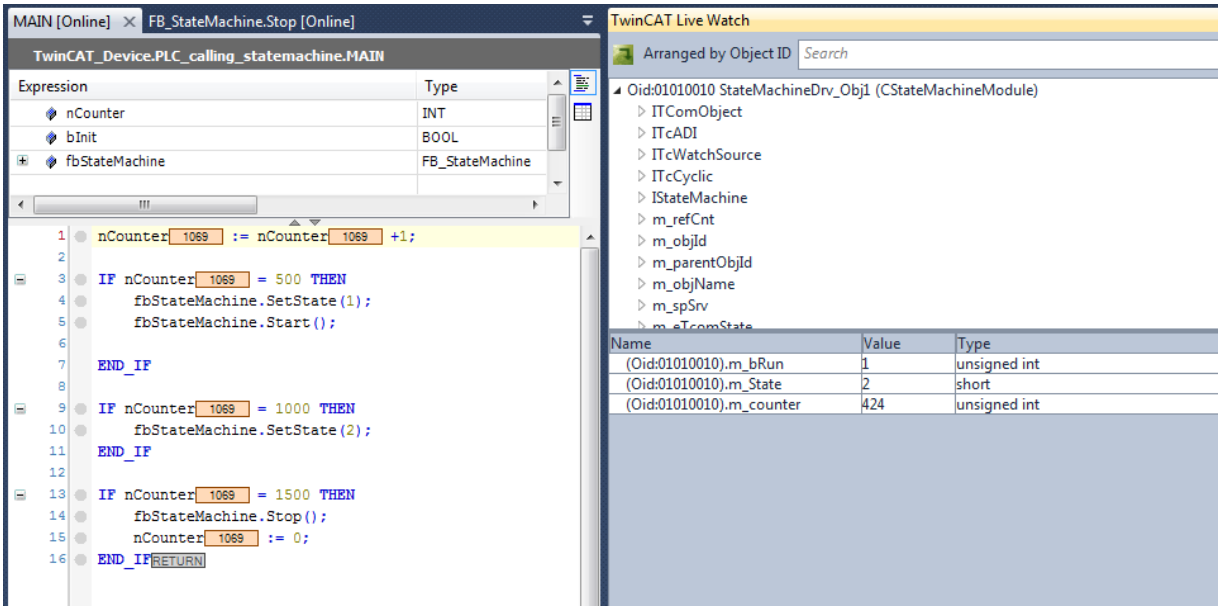


Schritt 10: Die Ausführung von beiden Modulen, SPS und C++, beobachten

Nach der Aktivierung der TwinCAT Konfiguration, dem Herunterladen und Starten des SPS-Codes, kann die Ausführung der beiden Codes, SPS und C++, einfach beobachtet werden:

1. Nach dem „Login“ und „Start“ des SPS-Projekts, befindet sich der Editor bereits im online-Modus (linke Seite der folgenden Abbildung).

2. Um auf online-Variablen des C++ Moduls zugreifen zu können, aktivieren Sie bitte das C++ Debugging [▶ 51] und folgen den Schritten des Schnellstarts [▶ 37], um das Debugging zu starten (rechte Seite der folgenden Abbildung).



15.10 Beispiel11a: Modulkommunikation: C-Modul führt eine Methode in C-Modul an

Dieser Artikel beschreibt, wie TC3 C++ Module über Methodenaufrufe kommunizieren können. Die Methode schützt die Daten durch eine Critical Section, folglich kann der Zugriff von verschiedenen Kontexten / Tasks initiiert werden.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Das Projekt enthält drei Module:

- Die Instanz der CModuleDataProvider Klasse hostet die Daten und schützt vor dem Zugriff mit Hilfe der „Retrieve“ und „Store“ Methoden durch eine Critical Section.
- Die Instanz der Modulklasse „CModuleDataRead“ liest die Daten aus dem DataProvider über den Aufruf der Retrieve Methode.
- Die Instanz der Modulklasse „CModuleDataWrite“ schreibt die Daten aus dem DataProvider über den Aufruf der Store Methode.

Die Read/Write Instanzen werden für den Zugriff auf die DataProvider Instanz konfiguriert, die im Menü „Interface Pointer“ auf der Instanzkonfiguration zu sehen ist.
Dort kann man auch den Kontext (Task) konfigurieren, in dem die Instanzen auszuführen sind. Bei diesem Beispiel werden zwei Tasks verwendet, TaskRead und TaskWrite.

Die „DataWriteCounterModulo“ Parameter von „CModuleDataWrite“ und auch DataReadCounterModulo („CModuleDataRead“) ermöglichen die Bestimmung des Moments, an dem die Modulinstanzen den Zugriff initiieren.

15.11 Beispiel12: Modulkommunikation: Verwendet IO Mapping

Dieser Artikel beschreibt, wie zwei TC3 C++ Module über das gewöhnliche IO Mapping von TwinCAT 3 kommunizieren können: Zwei Instanzen sind über das IO Mapping verknüpft und greifen periodisch auf den Variablenwert zu.

Source

Erhalten Sie den Quellcode für dieses Beispiel:

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Beide Instanzen werden anhand einer Modulklasse „ModuleInToOut“ realisiert: Die Klasse kopiert zyklisch

ihren Eingangsdatenbereich „Value“ auf ihren Ausgangsdatenbereich „Value“.

Die Instanz „Front“ agiert als Frontend für den Nutzer. Ein Eingangs-„Value“ wird - durch die Methode cycleupdate() - an den Ausgangs-„Value“ übergeben.

Dieser Ausgangs-„Value“ von „Front“ wird dem Eingangs-„Value“ der Instanz „Back“ zugeordnet (verknüpft).

Die „Back“ Instanz wiederum kopiert den Eingangs-„Value“ auf ihren Ausgangs-„Value“, was vom Nutzer beobachtet werden kann (siehe folgende Schritte des Schnellstarts um das Debugging zu starten: TwinCAT 3 C++ Projekt debuggen [▶ 53])

Letztendlich kann der Nutzer den Eingangs-„Value“ der „Front“ Instanz festlegen und den Ausgangs-„Value“ von "Back" beobachten.

15.12 Beispiel19: Synchroner Dateizugriff

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Moduls, das auf Dateien auf der Festplatte während des Starts eines Moduls, also in Echtzeitumgebung, zugreifen kann.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Der gesamte Quellcode, der vom Assistenten nicht automatisch generiert wird, wird mit dem Kommentarbeginn-Flag „//sample code“ und dem Kommentarende-Flag „//sample code end“ gekennzeichnet.
Somit können Sie nach diesen Zeichenketten in den Dateien suchen, um sich ein Bild von den Einzelheiten zu machen.

Beschreibung

Dieses Beispiel beschreibt den Zugriff auf Dateien über die TwinCAT Schnittstelle „ITCFileAccess“. Der Zugriff ist synchron und kann z.B. für das Lesen einer Konfiguration beim Starten eines Moduls verwendet werden.

Das Beispiel beinhaltet ein C++ Modul „TcFileTestDrv“ mit einer Instanz von diesem Modul „TcFileTestDrv_Obj1“.
Bei diesem Beispiel findet der Dateizugriff beim Übergang „PREOP to SAFEOP“ statt, also in der „SetObjStatePS()“ Methode.

Hilfsmethoden kapseln den Umgang mit Dateien ein.
Zuerst wird als Beispiel eine allgemeine Dateiinformation und eine Verzeichnisliste in die Protokollvorrichtung von TwinCAT 3 gedruckt. Anschließend wird eine Datei „%TC_TARGETPATH %DefaultConfig.xml“ (normalerweise „C:\TwinCAT\3.x\Target\DefaultConfig.xml“) auf „%TC_TARGETPATH %DefaultConfig.xml.bak“ kopiert.

Zwecks Zugang zu den Protokolleinträgen, siehe Registerkarte „Error List“ vom TwinCAT 3 Ausgabefenster. Sie können die Informationsmenge mittels Änderung der Variablen TraceLevelMax in der Instanz „TcFileTestDrv_obj1“ in Registerkarte „Parameter (Init)“ einstellen.

15.13 Beispiel20: FileIO-Write

Dieser Artikel beschreibt die Implementierung von TC3 C++ Modulen, die (Prozess)Werte in eine Datei schreiben.

Das Beschreiben der Datei wird von einem deterministischen Zyklus veranlasst - die Ausführung von File IO ist entkoppelt (asynchron), d.h.: Der deterministische Zyklus läuft weiter und wird nicht durch das Schreiben in der Datei behindert.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Das Beispiel beinhaltet eine Instanz von „TcAsyncWritingModule“, die Daten in die Datei „AsyncTest.txt“ im Verzeichnis BOOTPRJPATH (normalerweise C:\TwinCAT\3.x\Boot) schreibt.

TcAsyncBufferWritingModule hat zwei Puffer (m_Buffer1, m_Buffer2), die abwechselnd mit aktuellen Daten gefüllt werden. Die Membervariable m_pBufferFill zeigt auf den derzeit zu füllenden Puffer. Wenn ein Puffer vollständig gefüllt ist, dann wird die Membervariable m_pBufferWrite so gesetzt, dass sie auf den vollen Puffer zeigt.

Diese Daten werden mit Hilfe von TcFsmFileWriter in eine Datei geschrieben.

Beachten Sie, dass die Datei keinen von Menschen lesbaren Inhalt, wie ASCII Zeichen enthält, stattdessen werden in diesem Beispiel binäre Daten in die Datei geschrieben.

15.14 Beispiel20a: FileIO-Cyclic Read / Write

Dieser Artikel stellt ein umfassenderes Beispiel als S20 und S19 dar. Es beschreibt zyklischen Lese- und/oder Schreibzugriff auf Dateien von einem TC3-C++ Modul aus.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Das Beispiel beschreibt den Zugriff auf Dateien für Lesen und/oder Schreiben über die Methode CycleUpdate, also auf zyklische Weise.

Dieses Beispiel beinhaltet die folgenden Projekte und Modulinstanzen.

- Eine statische Bibliothek (TcAsyncFileIo) bietet den Dateizugriff.
  Der Code für den Dateizugriff kann geteilt werden, also befindet sich dieser Code in einer statischen Bibliothek, die von den Treiberprojekten genutzt wird.
- Ein Treiber (TcAsyncBufferReadingDrv) stellt zwei Instanzen zur Verfügung
  - ReadingModule: Verwendet die statische Bibliothek, um die Datei AsyncTest.txt zu lesen
  - WriteDetectModule: Schreiboperationen erkennen und Leseoperationen veranlassen
- Ein Treiber (TcAsyncBufferWritingDrv) stellt eine Instanz zur Verfügung
  - WriteModule: Verwendet die statische Bibliothek, um die Datei AsyncTest.txt zu schreiben

Beim Starten des Beispiels beginnt das Schreibmodul mit dem Schreiben von Daten in die Datei, die sich im Bootprojekt-Pfad (normalerweise C:\TwinCAT\3.x\Boot\AsyncTest.txt) befindet. Die Eingangsvariable „bDisableWriting“ kann für die Verhinderung des Schreibens verwendet werden.

Die Objekte sind miteinander verbunden: Wenn das Schreiben erledigt ist, triggert das WritingModule das DetectModule von TcAsyncBufferReadingDrv. Dadurch wird ein Lesevorgang durch das ReadingModule veranlasst.

Bitte die „nBytesWritten“ / „nBytesRead“ Ausgangsvariablen des WritingModules / ReadingModules beobachten. Darüber hinaus werden Protokollnachrichten auf Ebene „verbose“ generiert. Diese können wie gehabt mit Hilfe des TraceLevelMax Parameters der Module konfiguriert werden.

- Ein Treiber (TcAsyncFileFindDrv) stellt eine Instanz zur Verfügung
  - FileFindModule: Listen Sie die Dateien eines Verzeichnisses mit Hilfe der statischen Bibliothek auf.

Lösen Sie die Aktion mit Hilfe der Eingangsvariablen „bExecute“ aus. Der Parameter „FilePath“ beinhaltet das Verzeichnis, dessen Dateien aufzulisten sind (Standardwert: „c:\TwinCAT\3.1\Boot*“).

Bitte beobachten Sie die Ablaufverfolgung (Protokollebene „Verbose“) bezüglich der Liste der gefundenen Dateien.

Das Beispiel verstehen

Das Projekt TcAsyncFileIO beinhaltet verschiedene, in einer statischen Bibliothek befindliche Klassen. Diese Bibliothek wird von Treiberprojekten für das Lesen und Schreiben verwendet.

Jede Klasse ist für eine Dateizugriffsoperation wie Öffnen / Lesen / Schreiben / Auflisten / Schließen / ... bestimmt. Da die Ausführung innerhalb eines zyklischen Echtzeitkontextes stattfindet, hat jede Operation einen Status und die Klasse kapselt diese Zustandsmaschine ein.

Als Einstiegspunkt zum Verständnis des Dateizugriffs, beginnen Sie bitte mit den TcFsmFileReader und TcFsmFileWriter Klassen.

Sollten zu viele Verlaufsverfolgungsmeldungen auftreten, die das Verständnis des Beispiels erschweren, deaktivieren Sie Module!

Siehe auch

Beispiel S19 [▶ 248]

Beispiel S20 [▶ 249]

Beispiel S25 [▶ 252]

Schnittstelle ITcFileAccess [▶ 133] / Schnittstelle ITcFileAccessAsync [▶ 141]

Sehen Sie dazu auch

- Beispiel19: Synchroner Dateizugriff [▶ 248]
- Beispiel20: FileIO-Write [▶ 249]
- Beispiel25: Statische Bibliothek [▶ 252]

15.15 Beispiel22: Automation Device Driver (ADD): Zugang DPRAM

Dieser Artikel beschreibt die Implementierung eines TC3 C++ Treibers, der als TwinCAT Automation Device Driver (ADD) mit Zugriff auf DPRAM fungiert.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. **Konfigurationsdetails unten lesen** vor Aktivierung

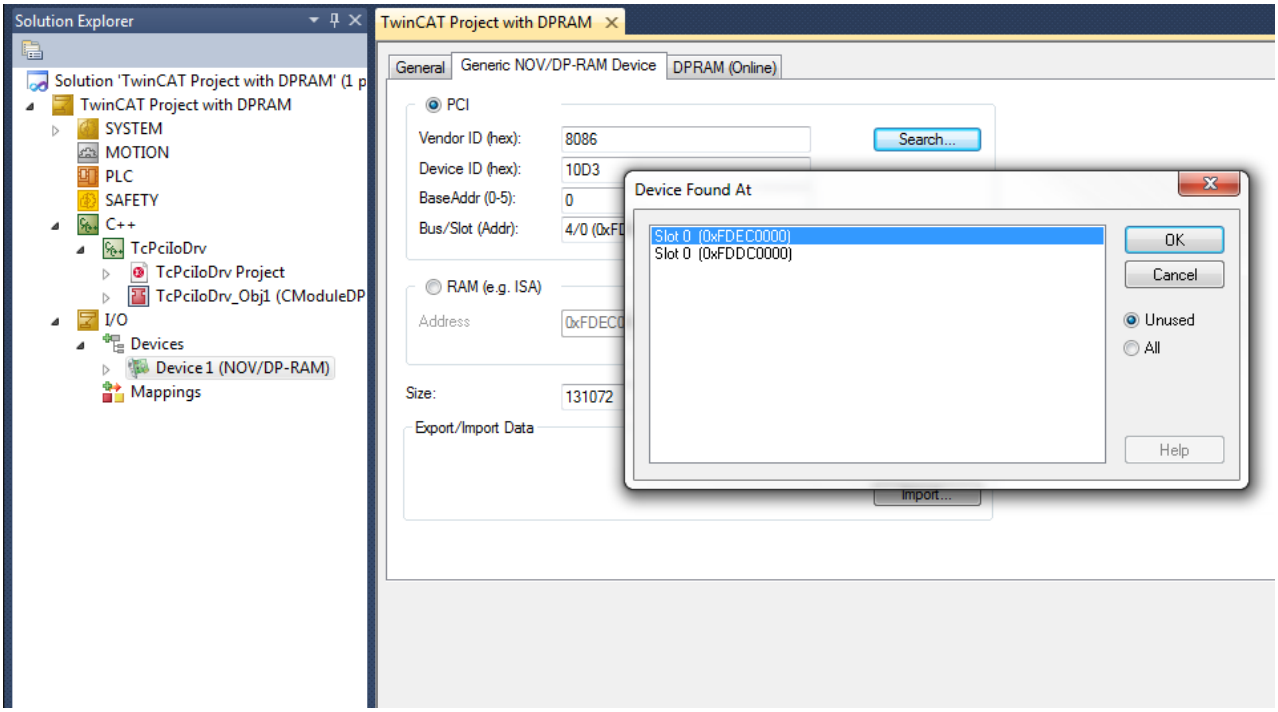
Beschreibung

Dieses Beispiel soll den „Link Detect Bit“ des Netzwerkadapters (d.h. von einem CX5010) zyklisch ein- und ausschalten.

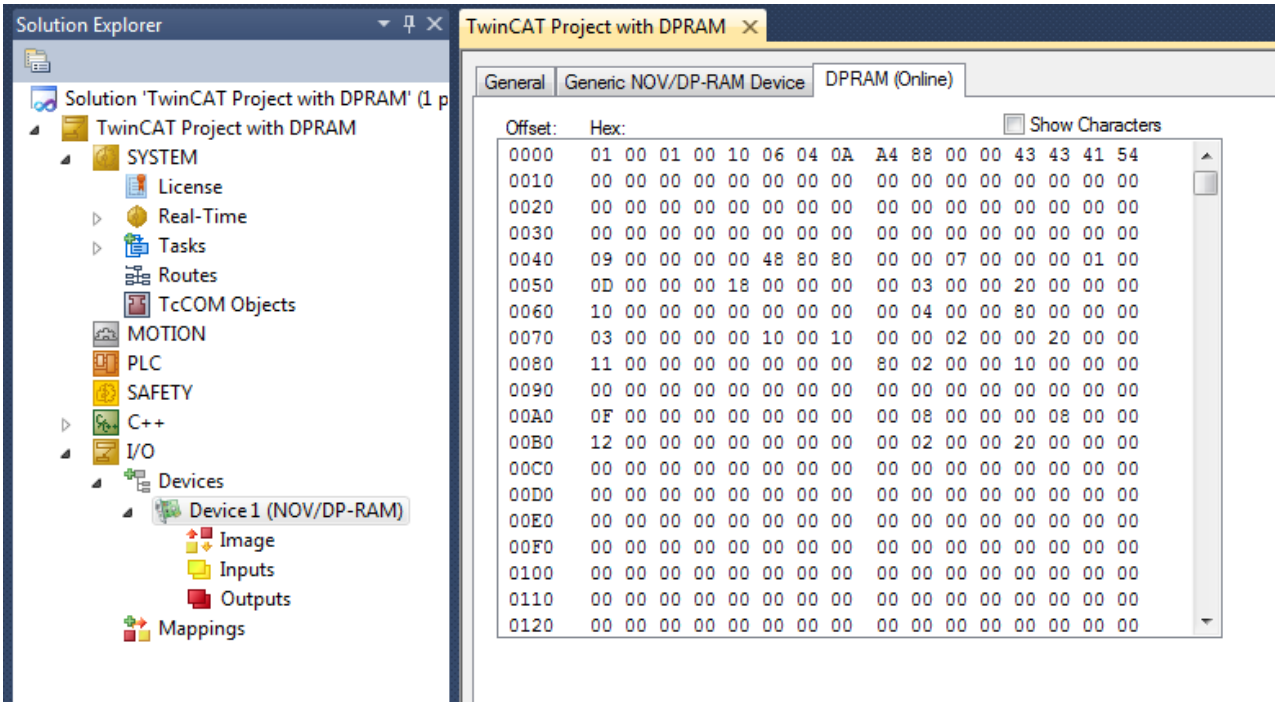
Das C++ Modul ist über den Schnittstellenzeiger „PciDeviceAdi“ des C++ Moduls mit dem NOV/DP-RAM Gerät verbunden.

Konfiguration

Damit das Beispiel funktioniert, müssen Sie die Hardware-Adressen passend zu Ihrer persönlichen Hardware konfigurieren.
Überprüfen Sie bitte die PCI Konfiguration:



Um zu überprüfen, ob die Kommunikation mit NOV/DPRAM korrekt eingerichtet ist, kann die DPRAM (Online) Ansicht verwendet werden:



15.16 Beispiel25: Statische Bibliothek

Dieser Artikel beschreibt die Implementierung und die Verwendung eines Moduls einer statischen TC3 C++ Bibliothek.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken

2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Das Beispiel beinhaltet zwei Projekte: Das Projekt „DriverUsingStaticLib“ verwendet den statischen Inhalt des Projekts „StaticLib“.

StaticLib: Auf der einen Seite bietet „StaticLib“ eine Funktion „ComputeSomething“ in der StaticFunction.h/.cpp.
Auf der anderen Seite wird eine Schnittstelle „ISampleInterface“ definiert (siehe TMCEditor) und in die MultiplicationClass implementiert.

DriverUsingStaticLib: In der CycleUpdate Methode der „ModuleUsingStaticLib“ wird sowohl die Klasse als auch die Funktion von „StaticLib“ verwendet.

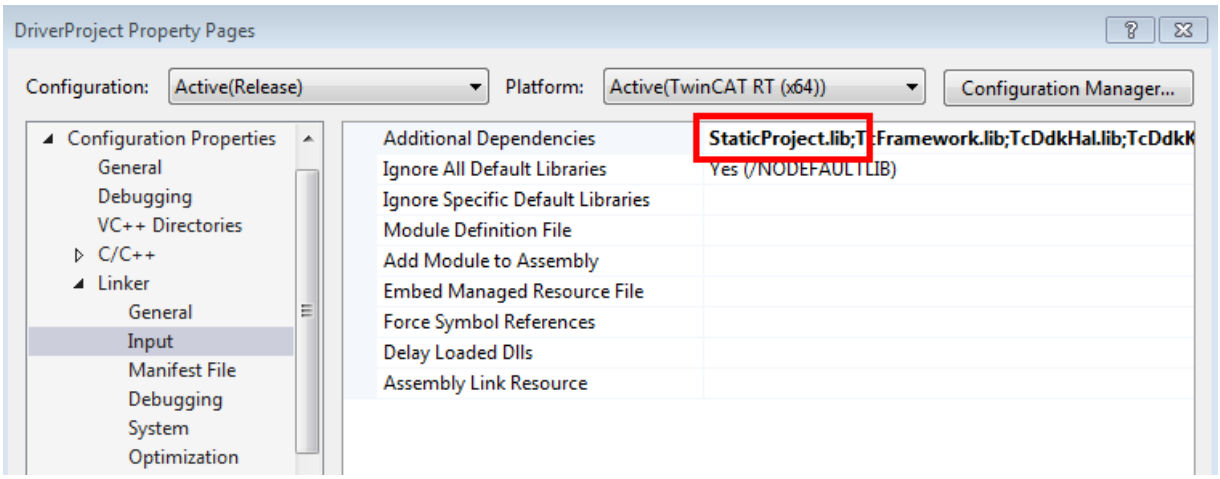
Das Beispiel verstehen

Durchlaufen Sie bitte folgende Schritte, um eine statische Bibliothek zu erstellen und zu verwenden.

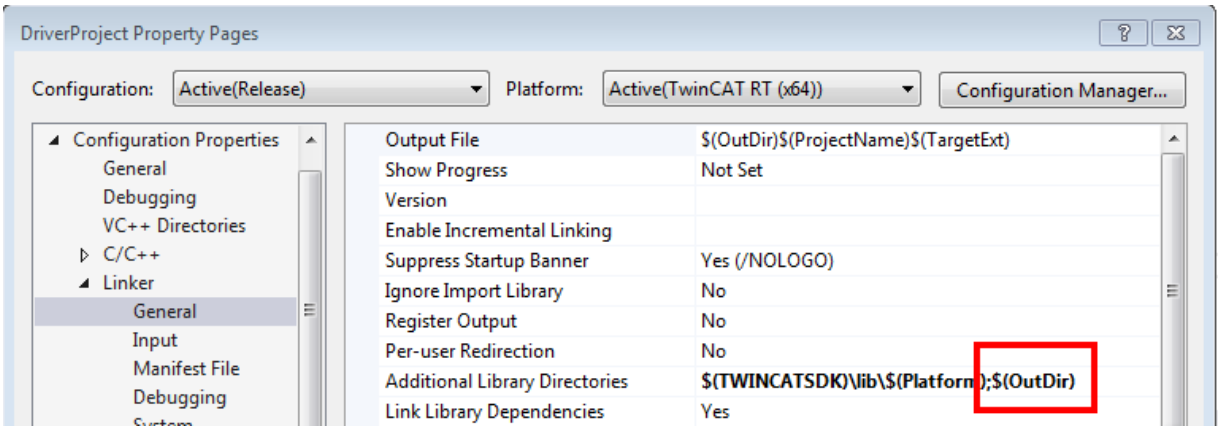
Hinweis	**Manuelle Neukompilierung** Beachten Sie, dass Visual Studio die statische Bibliothek bei der Erstellung des Treibers nicht automatisch erneut kompiliert. Führen Sie das bitte manuell aus.

✓ Nutzen Sie bei der Erstellung eines C++ Projekts bitte die „TwinCAT Static Library Project“ Vorlage für die Erstellung einer statischen Bibliothek.

6. Fügen Sie bitte im Treiber, der die statische Bibliothek nutzt, diese als eine zusätzliche Abhängigkeit für den Linker hinzu. Öffnen Sie bitte die Projekteigenschaften des Treibers und fügen die statische Bibliothek hinzu:



7. Darüber hinaus muss das Bibliotheksverzeichnis erweitert werden, damit der Linker die zusätzliche Bibliothek auch findet:



15.17 Beispiel26: Ausführungsreihenfolge in einem Task

Dieser Artikel beschreibt die Bestimmung der Taskausführungsreihenfolge, wenn einem Task mehr als ein Modul zugeordnet ist.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Das Beispiel enthält ein Modul „SortOrderModule“ das zwei Mal instanziiert wird. Die „Sortierreihenfolge“ bestimmt die Ausführungsreihenfolge, die über den TwinCAT Module Instance Configurator [▶ 112] konfiguriert werden kann.

Zum Beispiel verfolgt die „CycleUpdate“ Methode den Objekt-Namen und die Objekt-ID zusammen mit der Sortierreihenfolge dieses Moduls. Auf dem Konsolenfenster kann man die Ausführungsreihenfolge sehen:

```
32 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
33 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
34 05.09.2014 13:01:15 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
35 05.09.2014 13:01:15 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
36 05.09.2014 13:01:16 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
37 05.09.2014 13:01:16 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
```

Im Beispiel ist eine Instanz mit Sort Order 150 und eine mit 170 konfiguriert, während beide Instanzen einem Task zugeordnet sind.

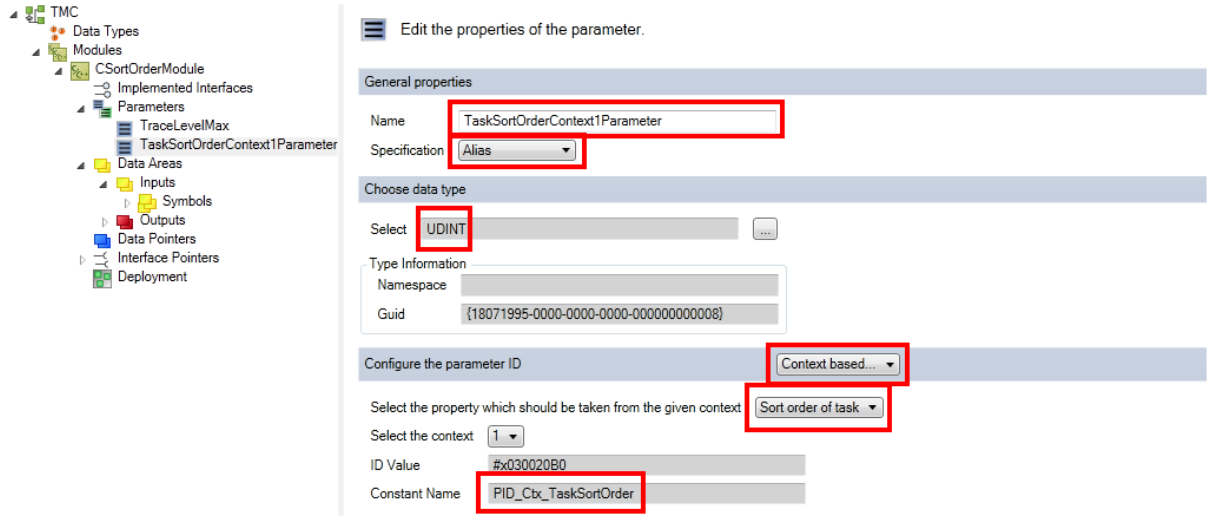
Startup

Das Beispiel benötigt nach dem Entpacken keine Konfiguration.

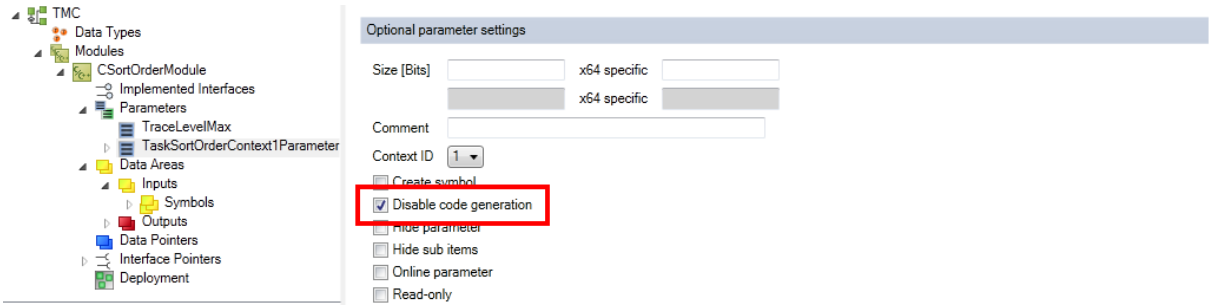
Das Beispiel verstehen

Annahme: Ein mit TwinCAT Module Klassenassistent [▶ 68] erstelltes Modul

6. Das Modul benötigt einen kontextbasierten Parameter „Sort order of task“, der automatisch „PID_Ctx_TaskSortOrder“ als Namen auswählen wird.
   Beachten Sie, dass der Parameter ein Alias (Spezifikation) zum Datentyp UDINT sein muss:



7. Bitten starten Sie den TMC Code Generator um die Standardimplementierung zu erhalten.
8. Da der Code im nächsten Schritt geändert werden wird, deaktivieren Sie bitte die Kodegenerierung für diesen Parameter jetzt.



9. Vergewissern Sie sich, dass Sie die Änderungen übernehmen, bevor Sie den TMC Code Generator erneut starten:
   Werfen Sie bitte einen Blick auf das CPP Modul (SortOrderModule.cpp im Beispiel). Die Instanz des Smart Pointers des zyklischen Aufrufers beinhaltet Informationsdaten, zu denen ein Feld für die Sortierreihenfolge gehört. In diesem Feld wird der Parameterwert gespeichert.

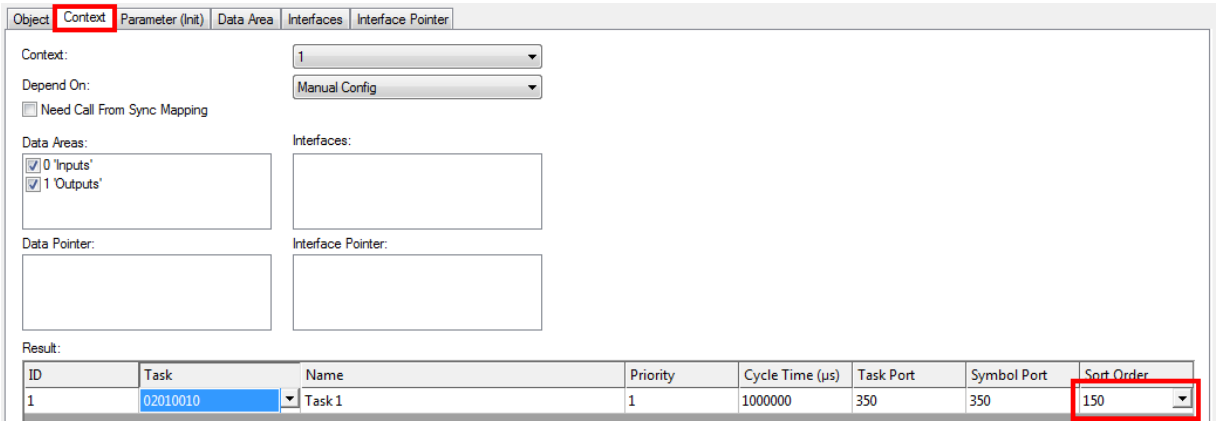
```
////////////////////////////////////////////////////////////////////////////////
// Set parameters of CSortOrderModule
BEGIN_SETOBJPARA_MAP(CSortOrderModule)
    SETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="SetObjectParameterMap">
    SETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    SETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
    SETOBJPARA_TYPE_CODE(PID_Ctx_TaskSortOrder, ULONG, m_spCyclicCaller.GetInfo()->sortOr-
der=*p) //ADDED
    //generated code: SETOBJPARA_VALUE(PID_Ctx_TaskSortOrder, m_TaskSortOrderContext1Parameter)
END_SETOBJPARA_MAP()

////////////////////////////////////////////////////////////////////////////////
// Get parameters of CSortOrderModule
BEGIN_GETOBJPARA_MAP(CSortOrderModule)
    GETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="GetObjectParameterMap">
    GETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    GETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
    GETOBJPARA_TYPE_CODE(PID_Ctx_TaskSortOrder, ULONG, *p=m_spCyclicCaller.GetInfo()->sortOr-
```

```
der) //ADDED
    //generated code: GETOBJPARA_VALUE(PID_Ctx_TaskSortOrder, m_TaskSortOrderContext1Parameter)
END_GETOBJPARA_MAP()
```

10. In diesem Beispiel werden Objekt-Name, -Id und SortOrder zyklisch verfolgt:

```
    // TODO: Add your cyclic code here
    m_counter+=m_Inputs.Value;
    m_Outputs.Value=m_counter;
    m_Trace.Log(tlAlways, FNAMEA "I am '%s' (0x%08x) w/ SortOrder %d ", this->TcGetObjectNa-
me(), this->TcGetObjectId() , m_spCyclicCaller.GetInfo()->sortOrder); //ADDED
```

11. Die Sortierreihenfolge kann auch als 4. Parameter der Methode ITcCyclicCaller::AddModule() übergeben werden, die in CModuleA::AddModuleToCaller() verwendet wird.
12. Den Instanzen dieses Moduls bitte einen Task mit **kleinem Zyklusintervall** (z.B. 1000ms) zuweisen, um die Verfolgungsmeldungen an das TwinCAT Engineering System zu begrenzen.
13. Jeder Instanz eine andere „Sortierreihenfolge“ über den TwinCAT Module Instance Configurator [▶ 112] zuweisen:



15.18 Beispiel30: Zeitmessung

In diesem Artikel wird die Implementierung eines TC3 C++ Moduls, das Zeitmessungsfunktionalitäten beinhaltet, beschrieben.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Dieses Beispiel befasst sich ausschließlich mit Zeitmessung wie

- Abfrage der Taskzyklenzeit in Nanosekunden
- Abfrage der Taskpriorität
- Abfrage der Zeit bei Taskzyklusstart in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).
- Abfrage der Distributed-Clock-Zeit bei Taskzyklusbeginn in Nanosekunden seit dem 1. Januar 2000.
- Abfrage der Zeit bei Methodenaufruf in Intervallen von 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Siehe auch

ITcTask Schnittstelle [▶ 150]

15.19 Beispiel31: Funktionsbaustein TON in TwinCAT3 C++

Dieser Artikel beschreibt die Implementierung eines Verhaltens in C++, das vergleichbar mit einem TON Funktionsbaustein von SPS / 61131 ist.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Das Verhalten dieses Moduls ist vergleichbar mit einem Modul, das mit dem „Cyclic IO“ Assistenten erstellt wurde. m_input.Value wird dem m_Output.Value hinzugefügt. Im Gegensatz zu „Cyclic IO“ Modul, fügt dieses Modul lediglich m_input.Value zu m_Output.Value hinzu, wenn die definierte Zeitspanne (1000ms) abgelaufen ist.

Dies wird mit Hilfe einer CTON Klasse erzielt, die mit dem TON Funktionsbaustein von SPS / 61131 vergleichbar ist.

Das Beispiel verstehen

Die C++ Klasse CTON (TON.h/.cpp) stellt das Verhalten eines TON Funktionsbausteins von SPS / 61131 zur Verfügung. Die Methode Update() ist vergleichbar mit dem Rumpf des Funktionsbausteins, der regelmäßig aufgerufen werden muss.

Die Methode Update() erhält zwei „in“ Parameter:

- IN1: Startet die Zeitschaltuhr mit steigender Flanke, setzt die Zeitschaltuhr mit fallender Flanke zurück
- PT: Abzuwartende Zeit, bevor Q gesetzt wird

Und zwei „out“ Parameter:

- Q: ist TRUE, wies PT Sekunden nach IN eine steigende Flanke auf
- ET: abgelaufene Zeit

Darüber hinaus muss ITcTask zur Abfrage der Zeitbasis bereitgestellt werden.

Siehe auch

Beispiel30: Zeitmessung [▶ 256]

ITcTask Schnittstelle [▶ 150]

15.20 Beispiel35: Ethernet Zugriff

Dieser Artikel beschreibt die Implementierung von TC3 C++ Modulen, die direkt über eine Ethernet Karte kommunizieren. Der Beispielcode fragt eine Hardware-Adresse (MAC) von einem Kommunikationspartner mittels zyklischem Senden und Empfangen von ARP-Paketen ab.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. **Folgen Sie den „Configuration“ Schritten unten**
6. Die Konfiguration aktivieren

Beschreibung

Das Beispiel beinhaltet eine Instanz des „TcEthernetSample“ Moduls, das ARP-Pakete zwecks Bestimmung der fernen Hardware-Adresse (MAC) sendet und empfängt.

Die CycleUpdate Methode implementiert eine rudimentäre Zustandsmaschine für das Versenden von ARP-Paketen und das Warten auf eine Antwort mit einer Zeitüberschreitung.

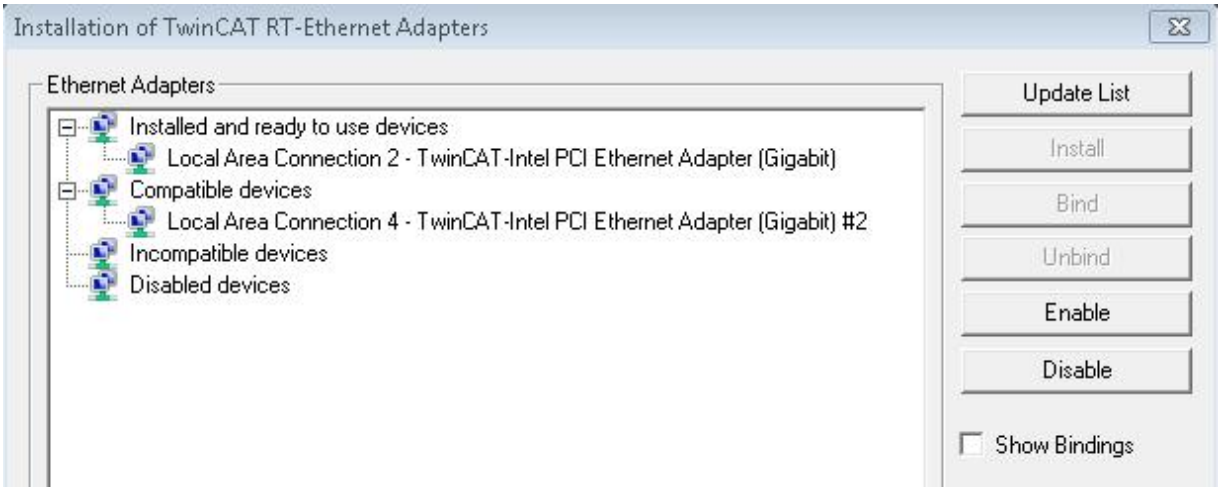
Das Beispiel verwendet zwei Ethernet-Komponenten von TwinCAT:

1. Ein **ITcEthernetAdapter** (Instanzenname in Beispiel ist m_spEthernetAdapter) stellt einen RT Ethernet Adapter dar. Er ermöglicht den Zugriff auf die Adapterparameter wie Hardware-MAC-Adresse, Verknüpfungsgeschwindigkeit, Verknüpfungsfehler. Er kann für das Senden von Ethernet-Frames verwendet werden und ermöglicht einer Modulinstanz, sich als ein ITcIoEthProtocol über die Methode registerProtocol anzumelden.
2. Das **ITcIOoEthProtocol** wird um das Abtastmodul erweitert, das dafür sorgt, dass eine Notifizierung bei Ethernet-Ereignissen über den **ITcEthernetAdapter** stattfindet.

Konfiguration

Das heruntergeladene TwinCAT Projekt muss für die Ausführung in Netzwerkumgebung konfiguriert sein. Bitte führen Sie die folgenden Schritte aus:

7. Das Beispiel verlangt, dass die Ethernet-Karte den TwinCAT Treiber verwendet.
   Starten Sie „TcRteInstall.exe“ entweder vom XAE über das Menü TwinCAT->“Show Realtime Ethernet compatible devices…“ oder von der Festplatte auf den XAR Systemen.



8. Möglicherweise müssen Sie den Treiber mit Hilfe der Schaltflächen installieren und aktivieren.
9. TwinCAT muss wissen, welche Ethernet-Karte verwendet werden sollte. Öffnen Sie das Projekt in XAE und I/O / Devices / Device 1 (RT-Ethernet Adapter) auswählen.
10. Die zweite Registerkarte „Adapter“ auswählen und mit „Search“ den Adapter auswählen.

11. TcEthernetSample_Obj1 muss konfiguriert werden. Das Instanzenfenster öffnen und die folgenden Werte festlegen:
    Parameter (Init): SenderIpAddress (IP von in Schritt 2 konfiguriertem Netzwerkadapter)
    Parameter (Init): TargetIpAddress (IP von Ziel-Host)
    Schnittstellenzeiger: EthernetAdapter muss auf I/O / Devices / Device 1 (RT-Ethernet Adapter) zeigen

15.21 Beispiel37: Daten archivieren

Das Beispiel TcCOM Object Archiv beschreibt das Wiederherstellen und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung.

Source

Erhalten Sie den Quellcode für dieses Beispiel

1. Die heruntergeladene ZIP-Datei entpacken
2. Die enthaltene sln-Datei mit TwinCAT 3 öffnen
3. Ihr Zielsystem auswählen
4. Das **Beispiel auf Ihre lokale Maschine erstellen** (auf das C++ Projekt rechtsklicken und „Build“ klicken)
5. Die Konfiguration aktivieren

Beschreibung

Das Beispiel TcCOM Object Archiv beschreibt das Wiederherstellen und Speichern des Zustands eines Objekts während der Initialisierung und Deinitialisierung. Der Zustand der Beispielklasse CModuleArchive entspricht dem Wert des Zählers CModuleArchive::m_counter.

Beim Übergang von PREOP zu SAFEOP, d.h. beim Aufruf von Methode CModuleArchive::SetObjStatePS(), wird der Objektarchivserver (ITComObjArchiveServer) für die Erstellung eines Objektarchivs zum Lesen benutzt, auf das über die Schnittstelle ITComArchiveOp zugegriffen wird. Diese Schnittstelle stellt Überladungen von operator>>() zur Verfügung um im Archiv zu lesen.

Beim Übergang von SAFEOP zu PREOP, d.h. beim Aufruf von Methode CModuleArchive::SetObjStateSP(), wird der TCOM Objektarchivserver für die Erstellung eines Objektarchivs zum Schreiben benutzt, auf das über die Schnittstelle ITComArchiveOp zugegriffen wird. Diese Schnittstelle stellt Überladungen von operator<<() zur Verfügung um im Archiv zu schreiben.

16 Anhang

16.1 ADS Return Codes

Fehlercode: 0x000 [▶ 260]..., 0x500 [▶ 260]..., 0x700 [▶ 261]..., 0x1000 [▶ 263].....

Globale Fehlercodes

Hex	Dec	Beschreibung
0x0	0	Kein Fehler
0x1	1	Interner Fehler
0x2	2	Keine Echtzeit
0x3	3	Zuweisung gesperrt-Speicherfehler
0x4	4	Postfach voll
0x5	5	Falsches HMSG
0x6	6	Ziel-Port nicht gefunden
0x7	7	Zielrechner nicht gefunden
0x8	8	Unbekannte Befehl-ID
0x9	9	Ungültige Task-ID
0xA	10	Kein IO
0xB	11	Unbekannter ADS-Befehl
0xC	12	Win32 Fehler
0xD	13	Port nicht angeschlossen
0xE	14	Ungültige ADS-Länge
0xF	15	Ungültige AMS Net ID
0x10	16	niedrige Installationsebene
0x11	17	Kein Debugging verfügbar
0x12	18	Port deaktiviert
0x13	19	Port bereits verbunden
0x14	20	ADS Sync Win32 Fehler
0x15	21	ADS Sync Timeout
0x16	22	ADS Sync AMS Fehler
0x17	23	Keine Index-Map für ADS Sync vorhanden
0x18	24	Ungültiger ADS-Port
0x19	25	Kein Speicher
0x1A	26	TCP Sendefehler
0x1B	27	Host nicht erreichbar
0x1C	28	Ungültiges AMS Fragment

Router Fehlercodes

Hex	Dec	Name	Beschreibung
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht. Die aktuell gesendete Nachricht wurde abgewiesen.
0x503	1283	ROUTERERR_DEBUGBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht. Die gesendete Nachricht wird nicht im ADS Monitor angezeigt.
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	ROUTERERR_PORTALREADYINUSE	Die gewünschte Portnummer ist bereits vergeben.
0x507	1287	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	ROUTERERR_NOMOREQUEUES	Die maximale Anzahl von Ports ist erreicht.
0x509	1289	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.

Allgemeine ADS Fehlercodes

Hex	Dec	Name	Beschreibung
0x700	1792	ADSERR_DEVICE_ERROR	Gerätefehler
0x701	1793	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt
0x702	1794	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe
0x703	1795	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset
0x704	1796	ADSERR_DEVICE_INVALIDACCESS	Lesen und schreiben nicht gestattet.
0x705	1797	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt
0x706	1798	ADSERR_DEVICE_INVALIDDATA	Ungültige Parameter-Werte
0x707	1799	ADSERR_DEVICE_NOTREADY	Gerät ist nicht betriebsbereit
0x708	1800	ADSERR_DEVICE_BUSY	Gerät ist beschäftigt
0x709	1801	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext (muss in Windows sein)
0x70A	1802	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher
0x70B	1803	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte
0x70C	1804	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...)
0x70D	1805	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl
0x70E	1806	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein
0x70F	1807	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden
0x710	1808	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden
0x711	1809	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig
0x712	1810	ADSERR_DEVICE_INVALIDSTATE	Gerät im ungültigen Zustand
0x713	1811	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt
0x714	1812	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig
0x715	1813	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert
0x716	1814	ADSERR_DEVICE_NOMOREHDLS	Keine weitere Notification Handle
0x717	1815	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß
0x718	1816	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert
0x719	1817	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout
0x71A	1818	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen
0x71B	1819	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert
0x71C	1820	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig
0x71D	1821	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig
0x71E	1822	ADSERR_DEVICE_PENDING	Anforderung steht aus
0x71F	1823	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen
0x720	1824	ADSERR_DEVICE_WARNING	Signal-Warnung
0x721	1825	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index
0x722	1826	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv
0x723	1827	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert
0x724	1828	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz
0x725	1829	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen
0x726	1830	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten
0x727	1831	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig
0x728	1832	ADSERR_DEVICE_LICENSESYSTEMID	Lizenz der System-ID ungültig
0x729	1833	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt
0x72A	1834	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft
0x72B	1835	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang
0x72c	1836	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart
0x72D	1837	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen
0x72E	1838	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur
0x72F	1839	ADSERR_DEVICE_CERTIFICATEINVALID	öffentliches Zertifikat
0x740	1856	ADSERR_CLIENT_ERROR	Clientfehler
0x741	1857	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter
0x742	1858	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer
0x743	1859	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz
0x744	1860	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung
0x745	1861	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten
0x746	1862	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem
0x747	1863	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert
0x748	1864	ADSERR_CLIENT_PORTNOTOPEN	ADS-Port nicht geöffnet
0x750	1872	ADSERR_CLIENT_NOAMSADDR	Interner Fehler in Ads-Sync

Hex	Dec	Name	Beschreibung
0x751	1873	ADSERR_CLIENT_SYNCINTERNAL	Hash-Tabelle-Überlauf
0x752	1874	ADSERR_CLIENT_ADDHASH	Schlüssel nicht gefunden im Hash
0x753	1875	ADSERR_CLIENT_REMOVEHASH	Keine weitere Symbole im Cache
0x754	1876	ADSERR_CLIENT_NOMORESYM	Ungültige Antwort empfangen
0x755	1877	ADSERR_CLIENT_SYNCRESINVALID	Sync Port ist gesperrt

RTime Fehlercodes

Hex	Dec	Name	Beschreibung
0x1000	4096	RTERR_INTERNAL	Interner Fehler im TwinCAT Echtzeit-System.
0x1001	4097	RTERR_BADTIMERPERIODS	Timer-Wert ist nicht gültig.
0x1002	4098	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	RTERR_INVALIDSTACKPTR	Task Stackpointer hat den ungültigen Wert 0.
0x1004	4100	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	RTERR_NOMORETCB	Kein freies TCB (Task Control Block) zur Verfügung. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	RTERR_EXTIRQNOTDEF	Kein externer Synchronisations-Interrupt angewandt.
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs- Interrupts ist fehlgeschlagen
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274c	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten. Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274d	10061	WSAECONNREFUSED	Verbindung abgelehnt. Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
			Weitere Winsock-Fehlercodes: Win32-Fehlercodes